AFRL-IF-RS-TR-2001-44
Final Technical Report
April 2001

MOBILE INFORMATION AGENTS

Dartmouth College

Daniela Rus

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2001-44 has been reviewed and is approved for publication.

APPROVED: Raymond A. Liuzzi

RAYMOND A. LIUZZI
Project Engineer

FOR THE DIRECTOR: James A. Collins

JAMES A. COLLINS, Acting Chief
Information Technology Division
Information Directorate

If your address has changed or if you wish to be removed from the Air Force Research Laboratory Rome Research Site mailing list, or if the addressee is no longer employed by your organization, please notify AFRL/IFTD, 525 Brooks Road, Rome, NY 13441-4505. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document require that it be returned.

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2001	Final Jan 98 - Jan 01

4. TITLE AND SUBTITLE
MOBILE INFORMATION AGENTS

5. FUNDING NUMBERS
C - F30602-98-C-0006
PE - 63728F
PR - 2532
TA - 01
WU - 52

6. AUTHOR(S)
Daniela Rus

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Department of Computer Science
Dartmouth College
Hanover NH 03755

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/IFTD
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2001-44

11. SUPPLEMENTARY NOTES
Air Force Research Laboratory Project Engineer: Raymond A. Liuzzi/IFTD/(315) 330-3577

12a. DISTRIBUTION AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*
The main goals for this project were to develop automated information organization algorithms, and to integrate the information organization algorithms in a mobile agent platform. The main objective was to investigate and demonstrate the value of a paradigm of computation in heterogeneous distributed systems with non-permanent network connections, in which mobile agents bring the computation to the data. As a result, a system called D'Agents that supports mobile agents has been developed. D'Agents is especially suited to distributed information access experiments in a network of mobile computers, such as laptops, palmtops, and other wireless devices. In addition, this effort has developed, implemented, and evaluated an information organization algorithm called the star algorithm. The star algorithm gives an organization of collection into clusters. Results for this effort include:

1. Information overload is a serious problem and efficient automatic information organization algorithms are useful in addressing this problem.
2. The Star Clustering algorithms: a. is the best performing algorithm for large-scale information organization, b. can be used in an on-line or off-line fashion and has several scalable extensions, c. has been analyzed and this effort's large-scale experiments math the theory, d. can be used for filtering applications and for persistent queries.
3. By combining the Star clustering algorithm with the power of mobile agent system, we increase system performance dramatically.

14. SUBJECT TERMS
Knowledge Base, Software, Data Base, Computers

15. NUMBER OF PAGES
144

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

1 Research Summary

1.1 Project goals

The main goals for our project have been:

1. to develop automated information organization algorithms
2. to integrate the information organization algorithms in a mobile agent platform

We have fulfilled both goals successfully. The following sections detail our results.

1.2 Background

Information in electronic form is proliferating rapidly in a variety of forms. We now have powerful search engines that can return pointers to millions of documents on any subject. How can we tap into this fortune, while avoiding information overload?

The productivity and success of an individual in a society innundated with electronic data will be largely determined by timely access to information. This is particularly challenging when the data is unstructured, active, and heterogeneous. It seems unlikely that we could package information in a standardized form for the purposes of extraction and interpretation, because people's information needs are varied and the production of information is easy. The production of information is such that "manufacturing" facilities can be moved easily at little, or no cost, giving rise to transient datasources. Just as the invention of the railroad (and other means of mass transportation) has made it possible for comsumers to obtain products quickly, our vision is to provide ubiquitous, customized, and organized access to all users. To this end, we advocate technologies for systems in which customers can express information needs in flexible ways, and that provide facilities for an intelligent and customized exploration of the Web and other information spaces.

To build useful tools for tapping into the vast evolving net of information resources, we need to address two fundamental issues.

a. Data access in modern computing environments: how do we access information in a computing environment defines by dynamic wireless compute platforms and transient databases?

Computers are departing from their traditional desk-top configurations and becoming more portable. We now have wireless computers and palm-top computers that can interface with the rest of the elctronic world independent of their physical location. In an environment with ubiquitous computers, we would like to provide ubiquitous access to computers and information. Sample applications include anywhere, anytime communication, flexible scheduling, smart rooms, embedded devices, support for collaborative decision making, etc. The key research questions that need to be addressed to support such applications include: (1) what is the hardware infrastructure best suited for this task?; (2) how can we provide active networking in a dynamic system of computers?; and (3) how can we locate users and forward information when the originating host might become disconnected?

b. Data evolution: how do we organize an ever-changing information space?

As information systems grow, they need to autonomously reorganize themselves to effectively meet requests for information. Such reorganization could involve simple processes like active selection and caching of information views that facilitate query processing, to more complex processes that generate and maintain active indexes into information. Sample applications include automatic capture and access tasks in digital libraries. Information organization algorithm can be used to organize any collection of documents in a customized way. Users can organize their email files. Large corporations can organize their manuals, news releases, and internal documents. Such a system can be used to create "Web Centers" and "Yellow Pages" automatically and provide users with better interfaces. The system can alo be used as a front end to a search engine. The key to supporting such applications is reasearch that would lead to efficient algorithms for information organization that theoretically well-grounded and creating flexible, modular, and customizable systems that use these algorithms.

To address these issues effectively, we need new ways of conceptualizing and communicating information needs. We believe that a good approach is a computational paradigm relying on customizable mobile information agents. By agents we mean autonomous decision-making programs that migrate from host to host under their own control. By customizable we mean programs that evolve automatically with changes in the information landscape, as well as programs that can be easily modified and assembled by users, according to their tasks. By information agents we mean dedicated program that can run sophisticated information capture, access, and organization algorithms.

Our main objective has been to investigate and demonstrate the value of a paradigm of computation in heterogeneous distributed systems with non-permanent network connections, in which mobile agents bring the computation to the data. A mobile agent is an automous program that can migrate from machine to machine in a heterogeneous network, at times and to places of its own choosing. This navigation autonomy is very powerful, and requires an agent to have substantial intelligence in making decisions and filtering information.

We have built a system called D'Agents that supports mobile agents. D'Agents is especially suited to distributed information access experiments in a network of mobile computers, such as laptops, palmtops, and other wireless devices. Mobile computers do not have a permanent connection into the network and are often disconnected for a long period of time. We focus on applications that require extensive data processing over distributed and transient databases in wireless networks. We look for algorithms that allow agents to retrieve and *organize* the relevant data as naturally occuring hierarchies of topics and subtopics.

Mobile agents provide a convenient, efficient, and intelligent paradigm for implementing distributed applications, especially in the context of wireless computing. First, by migrating to the location of an electronic resource, an agent can access the resource locally and eliminate costly data transfers over congested networks. This reduces network traffic, because it is often cheaper to send a small agent to a data source than to send all the intermediate data to the requesting site. Second, the agent does not require a permanent connection to the host machine (*e.g.*, the computer from where an agent is launched). This capability supports distributed information-processing applications on mobile computers. Third, the network-sensing capabilities enable agents to autonomously find the host computer, even when the host changes its geographical location. Fourth, the network software- and hardware-sensing capabilities permit transportable

agents to navigate adaptively. Fifth, our transportable agents can communicate with each other even when they do not know their specific locations in the network. Finally, agents have autonomy in decision making: by using feedback from visiting a site, they can independently modify the overall plan or refine ill-specified queries. When combined with communication, decision-making enables our agents to be negotiators. D'Agents supports negotiation through an infrastructure that supports transactions on electronic cash, arbitration on electronic cash transactions, and economic policies for resource control.

Although this contract has not supported the entire development of the D'Agents system, it has supported a small portion of it. The rest of the project was supported by Darpa, AFOSR, and ONR. We used the D'Agents system as a lpatform in evaluating some of the applications of our information organization algorithms described in the next section.

1.3 Information Organization

For this aspect of our work, are motivated by a long-term vision in which information systems can help leaders make decisions by collecting, filtering, updating, and presenting information quickly, accurately, and effectively. Information systems will compute the underlying topic-subtopic structure of dynamic textual databases. As new information comes into the database, the system will fuse it with the existing topic structure. The system will also be able to remove documents from the database.

Our work focuseson a paradigm for organizing data that can be used as a pre-processing step in a static information system or as a post-processing step on the specific documents retrieved by a query. As a pre-processor, this system assists users with deciding how to browse the corpus by highlighting relevant topics and irrelevant subtopics. Such clustered data is useful for narrowing down the corpus over which detailed queries can be formulated. As a post-processor, this system classifies the retrieved data into clusters that capture topic categories and subcategories.

We have developed, implemented, and evaluated an information organization algorithm called the star algorithm. The star algorithm gives an organization of a collection into clusters. Each level in the hierarchy is determined by a threshold for the minimum similarity between pairs of documents within a cluster at that particular level in the hierarchy. This method conveys the topic-subtopic structure of the corpus according to the similarity measure used.

The problem can be formulated by representing an information system by its similarity graph. A similarity graph is an undirected, weighted graph G = (V,E,w) where vertices in the graph correspond to documents and each weighted edge in the graph corresponds to a measure of similarity between two documents. We measure the similarity between two documents by using the cosine metric in the vector space model of the Smart information retrieval system. G is a complete graph with edges of varying weight. An organization of the graph that produces reliable clusters of similarity sigma (i.e., clusters where documents pairwise have similarities of at least sigma) can be obtained by performing a minimum clique cover of all edges whose weights are above the threshold sigma. Unfortunately, this approach is computationally intractable. For real corpora, these graphs can be very large. The clique cover problem is NP-complete, and it does not admit polynomial-time approximation algorithms. While we cannot perform a clique cover nor even approximate such a cover, we can instead cover our graph by dense subgraphs.

Specifically, we use star-shaped subgraphs. A star-shaped subgraph on m+1 vertices consists of a single star center and m satellite vertices, where there exist edges between the star center and each of the satellite vertices. The star-cover algorithm is provably accurate in that it produces dense clusters with provable guarantees on the pairwise similarity between cluster documents, and it can be quickly computed. The documents in each cluster are tightly inter-related and a minimum similarity distance between all the document pairs in the cluster is guaranteed. This resulting structure reflects the underlying topic structure of the data. A topic summary for each cluster is provided by the center of the underlying star for the cluster.

This approach has three nice features. First, by using star-shaped graphs to cover the similarity graph, we are guaranteed that all the documents in a cluster have the desired degree of similarity. Second, covering the edges of the graph allows vertices to belong to several clusters. Documents can be members of multiple clusters, which is a desirable feature when documents have multiple subthemes. Third, this algorithm can be iterated for a range of thresholds, effectively producing a hierarchical organization structure for the information system. Each level in the hierarchy summarizes the collection at a granularity provided by the threshold.

We have developed a prototype system for doing this task. We have experimented with this system and found that the precision-recall is higher than the precision-recall of other techniques such as the single link method, average-link method, and the k-means method. We are currently working on an on-line version of this system that could organize dynamically-changing document collections.

These algorithms can be used to create automatically knowledge bases. A set of raw documents are indexed to create information bases. By clustering an information base and summarizing the results of the organized collection we add a higher-level of knowledge into the database. This type of knowledge can be used to reduce information overload and have applications in a variety of tasks, such as customized filtering of information, topics detection and tracking in continuous information streams, collaborative decision making, etc.

The off-line and on-line star algorithms can be optimized further for better performance. Note that both versions of the algorithm rely on the existence of the similarity matrix. Similarity matrices can get very large: for a document set with n documents the similarity matrix is $O(n^2)$ space data structure. However, this operation, which takes $O(n^2)$ time to compute, is much more expensive than the basic cost of the star clustering algorithm, which is approximately $O(V+E)$ time. Thus, it is clear that the similarity matrix is a bottleneck. Computing this matrix is a one-time pre-processing operation. However, the data structure has to be available on a permanent basis. For these reasons, it is important to consider methods that improve on the similarity matrix bottleneck.

We have developed, implemeted, and started testing an extension that approximates the star algorithm by using sampling to compute the similarity matrix. The basic idea is to create a sample of the document collection that is much smaller than the actual collection. This sample can then be used to compute a complete Star Clustering, using the off-line star algorithm and the remaining documents can be inserted in the resulting structure. An additional optimization is to remove entirely the similarity matrix. The key information used by the star algorithm is the degree of the nodes in the thresholded similarity graph. This information can be represented in an array and it can be computed approximately, by sampling.

Another bottleneck for the star algorithm comes up in Internet applications, such as organizing data collected from various sited and databases by topic. Consider a task in which several databases are queried with the same question. The documents returned by these queries are to be fused and presented to the user in a coherent picture. One approach is to run the queries, download all documents, and organize the entire collection at the user site using the star algorithm. An alternative approach is to run the queries, organize the search results at the location of the database, and then merge these results on the user machine. This second alternative has several advantages: (1) the star algorithm can be run in parallel, which provides a speedup; (2) the document transfer operation can also be parallelized (note that if the number of documents is large and the network bandwidth is low, the cost of the transfer can be overwhelming); and (3) the local topic organizations can be viewed as a way of compressing the documents, can be used to generate the merged topics in the distributed collection, and can be transfered much faster than the actual documents to the user's machine.

For these reasons, we developed a third approximation of the star algorithm called *the distributed star* algorithm, which is useful especially when the document collection is very large. The distributed star algorithm provides parallelism and is based on a "divide and conquer" approach. The document collection is partitioned into several disjoint sets. The sets are clustered separately and the resulting clusters are then merged. We are currently implementing this distributed version of the information organization system with D'agents. We plan to use this integrated version of the system as an application on top of Serval, a large distributed database of

In another project, we started to investigate a new information-theoretic model for document retrieval and clustering. In this model, a collection of text documents (the "corpus") is first analyzed to determine a *probability model* for the terms within the corpus. Terms that appear infrequently have relatively low assigned probabilities, while terms that appear frequently have relatively high assigned probabilities. The *Shannon information* is then computed for each of the terms in the corpus—it is simply the length of the codeword (in bits) assigned to each term in the optimal encoding scheme for compressing or transmitting the corpus. The Shannon information can be efficiently computed from the corpus probability model.

Given a query in the form of a collection of keywords, we can perform document retrieval by determining the total number of bits that each document contains about the given keywords and returning relevant documents ranked according to this measure. For each document, this bit total can be calculated by summing, for each keyword, the product of its Shannon information times the frequency with which that keyword appears in the document (normalized in such a way that "short" documents and "long" documents are treated equally). Clustering can be achieved via an information-theoretic *similarity measure* which can be derived within this model. Essentially, the pairwise-similarity between two documents corresponds to the fraction of keyword bits that the two documents share in common.

We have implemented a system employing these ideas on a large corpus containing some 130,000 documents. So far our results are encouraging, both in terms of accuracy (the "quality" of retrieved documents) and efficiency (query retrieval on 100,000+ documents in a fraction of a second on a PC).ject, we are currently investigating a new information-theoretic model for document retrieval and clustering. In this model, a collection of text documents (the "corpus") is first analyzed to determine a *probability model* for the terms within the corpus. Terms that

appear infrequently have relatively low assigned probabilities, while terms that appear frequently have relatively high assigned probabilities. The *Shannon information* is then computed for each of the terms in the corpus—it is simply the length of the codeword (in bits) assigned to each term in the optimal encoding scheme for compressing or transmitting the corpus. The Shannon information can be efficiently computed from the corpus probability model.

Given a query in the form of a collection of keywords, we can perform document retrieval by determining the total number of bits that each document contains about the given keywords and returning relevant documents ranked according to this measure. For each document, this bit total can be calculated by summing, for each keyword, the product of its Shannon information times the frequency with which that keyword appears in the document (normalized in such a way that "short" documents and "long" documents are treated equally). Clustering can be achieved via an information-theoretic *similarity measure* which can be derived within this model. Essentially, the pairwise-similarity between two documents corresponds to the fraction of keyword bits that the two documents share in common.

2 Lessons Learned

This effort has uncovered some valuable lessons for the computer science community, for the airforce, and for the population at large.

1. Information overload is a serious problem and efficient automatic information organization algorithms are useful in addressing this problem.

2. The Star Clustering algorithm is the best performing algorithm for large-scale information organization.

3. The Star clustering algorithm can be used in an on-line or off-line fashinon and has several scalable extensions.

4. The Star clustering algorithm has been analyzed and our large-scale experiments match the theory.

5. The Star clustering algorithm can be used for filtering applications and for persistent queries.

6. By combining the Star clustering algorithm with the power of mobile agent system we increase system performance dramatically. Specifically, we conserve bandwith by transfering the code to the data, performing data processing at the site of the data, and bringing back only the relevant results. In addition, mobile agents support multiple queries without connecting the the home machine and thus contribute to the reduction of the total completion time of a job. Finally, mobile agents support disconnected queries, in low-latency wireless networks.

3 Students

The following students were supported on this contract:

- Ekaterina Pelekhov, PhD student, thesis defended in May 2000; expecting the final version.
- Mark Montague, PhD student, thesis expected in May 2001.
- Ken Yasuhara, undergraduate student, currently a PhD student in the computer science department at the University of Washington.

In addition to these students, profs. Jay Aslam, prof. David Kotz, and prof. Daniela Rus were also supported in part by this contract.

4 Software

We designed and implemented a mobile-agent system called D'Agents (see http://www.cs.dartmouth.edu/~agent/agenttcl.html). We have completed several releases of this system that has security mechanism for protecting machines from malicious agents and several additional capabilities for agents over the previous release. These versions support Agent Tcl, Agent Java, and Agent Scheme as programming languages.

We also designed and implemented a system that supports automated information organization in static and dynamic environments, filtering on a text stream and persistent queries. The system has a novel graphical user interface that projects the topic content of the corpus onto a 2-dimensional window, thus supporting intuitive browsing to cope with information overload. The information organization software is available from http://www.cs.dartmouth.edu/~rus/Software/info-org.tar.gz.

5 Papers

The following papers resulted as part of this project:

"Automatic Information Organization" (with J. Aslam and K. Pelekhov), in *Proceedings of the 2000 SSGCC* (book with CD ROM).

"Mobile agents: motivations, state of the art, and frontiers" (with G. Cybenko, R. Gray, and D. Kotz), in eds. J. Bradshaw, *Handbook of Agent Technologies*, MIT Press, 1999 (to appear).

"Generating, visualizing, and evaluating high-quality clusters for information organization" (with J. Aslam and K. Pelekhov), in *Proceedings of Principles of Digital Document Processing* eds. E. Munson, C. Nicholas, D. Wood, Lecture Notes in Computer Science 1481, Springer-Verlag 1998.

"Applications of clustering to filtering and persistent queries" (with J. Aslam and K. Pelekhov), in *Proceedings of CIKM 2000* (November 2000).

"Scalable Information Organization" (with J. Aslam and F. Reiss), in Proceedings of RIAO 2000 (Content-based information access) (April 2000).

"A practical clustering algorithm for static and dynamic information organization" (with J. Aslam and K. Pelekhov), in the 1999 Symposium on Discrete Algorithms (SODA99), Baltimore, MD (January 1999).

"Static and Dynamic Information Organization with Star Clusters" (with J. Aslam and K. Pelekhov), In *Proceedings of the 1998 Conference on Intelligent Knowledge Management*, Washington, DC (November 1998).

6 Talks

"Mobile Information Agents", D. Rus, Caterpillar, Peoria, IL, December 1998.

"Mobile Information Agents", D. Rus, Rome Labs, December 1998.

"Mobile Information Agents", D. Rus, Qualcom distinguished lecture, The University of California at San Diego, February 1999.

"Mobile Information Agents", panel on modern information technologies moderated by Joe Cavano, COMPSAC 99, October 99.

"Scalable Extensions of the Star Algorithm", F. Reiss, RIAO 2000, April 2000.

"Information Organization Algorithms", D. Rus, SGGRR 2000, L'Aquila Italy, July 2000.

"Using the star clustering algorithm for filtering", J. Aslam, CIKM 2000, November 2000.

"D'Agents: a secure mobile agent system", D. Rus, NRL, December 2000.

7 Service to the Community

- D. Rus, Treasurer, The 1999 International Conference on Autonomous Agents
- D. Rus, Program Committee, The Workshop on Mobile Agents in the Context of Competition and Cooperation, 1999
- D. Kotz, Program Committee, The Workshop on Mobile Agents in the Context of Competition and Cooperation, 1999
- D. Rus, Senior Program Committee, 1999 International Joint Conference on Artificial Intelligence (IJCAI99)
- D. Rus, General Chair, Dartmouth Workshop on Mobile Agents, 1999, 2000
- D. Rus, Program Committee SIGIR
- D. Rus, Senior Program Committee, 2001 International Conference on Autonomous Agents

8 Interactions with other Agencies

We are working with Darpa as part of the Co-Abs project and with the Air Force as part of a MURI project. We have integrated the information organization system we developed as part of this contract in our MURI demo and hope to do some integration with the Darpa Grid and perhaps a Fleet Battle Experiment. We are looking for more venues to transition this work.

Static and Dynamic Information Organization with Star Clusters

Javed Aslam Katya Pelekhov Daniela Rus

Department of Computer Science
Dartmouth College
Hanover, NH 03755

Abstract

In this paper we present a system for static and dynamic information organization and show our evaluations of this system on TREC data. We introduce the off-line and on-line star clustering algorithms for information organization. Our evaluation experiments show that the off-line star algorithm outperforms the single link and average link clustering algorithms. Since the star algorithm is also highly efficient and simple to implement, we advocate its use for tasks that require clustering, such as information organization, browsing, filtering, routing, topic tracking, and new topic detection.

1 Introduction

Modern information systems have vast amounts of unorganized data that change dynamically. Consider, for example, the flow of information that arrives continuously on news wires, or is aggregated by a news organization such as CNN. Some stories are new while other stories are follow-ups on previous stories. Yet another type of stories make previous reportings obsolete. The news focus changes regularly with this flow of information. In such dynamic systems, users need to locate information quickly and efficiently.

Current information systems such as Inquery [Tur90], Smart [Sal91] and Alta Vista provide some simple automation by computing ranked (sorted) lists of documents, but it is ineffective for users to scan a list of hundreds of document titles. To cull the critical information out of a large set of potentially useful dynamic sources, we need methods for organizing information to highlight the topic content of a collection and reorganize the data to adapt to the incoming flow of documents. Such information organization algorithms would support incremental information processing tasks such as routing, topic tracking and new topic detection in a stream of documents.

In this paper, we present a system for the static and dynamic organization of information and we evaluate the system on TREC data. We introduce the off-line and on-line star clustering algorithms for information organization. We also describe a novel method for visualizing clusters, by embedding them in the plane so as to capture their relative difference in content. Our evaluation experiments show that the off-line star algorithm outperforms the single link and average link clustering algorithms. Since the star algorithm is also highly efficient and simple to implement, we advocate its use for tasks that require clustering, such as information organization, routing, topic tracking, and new topic detection.

1.1 Previous Work

There has been extensive research on clustering and applications to many domains [HS86, AB84]. For a good overview see [JD88]. For a good overview of using clustering in information retrieval see [Wil88].

The use of clustering in information retrieval was mostly driven by *the cluster hypothesis* [Rij79] which states that relevant documents tend to be more closely related to each other than to non-relevant documents. Efforts have been made to determine whether the cluster hypothesis is valid. Voorhees [Voo85] discusses a way of evaluating whether the cluster hypothesis holds and shows negative results. Croft [Cro80] describes a method for bottom-up cluster search that could be shown to outperform a full ranking system for the Cranfield collection. The single link method [Cro77] does not provide any guarantees for the topic similarity within a cluster. Jardine and van Rijsbergen [JR71] show some evidence that search results could be improved by clustering. Hearst and Pedersen [HP96] re-examine the cluster hypothesis by focusing on the Scatter/Gather system [CKP93] and conclude that it holds for browsing tasks.

Systems like Scatter/Gather [CKP93] provide a mechanism for user-driven organization of data into a fixed number of clusters, but user feedback is required and the computed clusters do not have accuracy guarantees. Scatter/Gather uses fractionation to compute nearest-neighbor clusters. In a recent paper, Charika et al. [CCFM97] consider a dynamic clustering algorithm to partition a collection of text documents into a fixed number of clusters. However, since the number of topics in a dynamic information systems is not generally known *a priori*, a fixed number of clusters cannot generate a natural partition of the information.

1.2 Our Work

Our work on clustering presented in this paper and in [APR98] describes a simple incremental algorithm, provides positive evidence for the cluster hypothesis, and shows promise for on-line tasks that require dynamically adjusting the topic content of a collection such as filtering, browsing, new topic detection and topic tracking. We propose an off-line algorithm for clustering static information and an on-line version of this algorithm for clustering dynamic information. These two algorithms compute clusters induced by the natural topic structure of the space. Thus, this work is different than [CKP93, CCFM97] in that we do not impose a fixed number of clusters as a constraint on the solution. As a result, we can guarantee a lower bound on the topic similarity between the documents in each cluster.

To compute accurate clusters, we formalize the clustering problem as one of covering a thresholded similarity graph by cliques. Covering by cliques is NP-complete and thus intractable for large document collections. Recent graph-theoretic results have shown that the problem cannot even be approximated in polynomial time [LY94, Zuc93]. We instead use a cover by dense subgraphs that are star-shaped[1], where the covering can be computed off-line for static data and on-line for dynamic data. We show that the off-line and on-line algorithms produce high-quality clusters very efficiently. Asymptotically, the running time of both algorithms is roughly linear in the size of the similarity graph that defines the information space. We also derive lower bounds on the topic similarity within clusters guaranteed by a star covering, thus providing theoretical evidence that the clusters produced by a star cover are of high-quality. We packaged these algorithms as a system that supports ad-hoc queries, static information organization, dynamic information organization, and routing. In this system we contributed a novel way of visualizing topic clusters by using disks whose radii are proportional to the size of the cluster and that are embedded in the plane in a way that captures the topic distance between the clusters. Finally, we provide experimental data for off-line and on-line topic organization. In particular, our off-line results on a TREC collection indicate that star covers exhibit significant performance improvements over either the single link [Cro77] or average link [Voo85] methods (21.6% and 16.2% improvements, respectively, with respect to a common cluster quality measure) without sacrificing simplicity or efficiency.

1.3 Utility

Our algorithms for organizing information systems can be used in several ways. The off-line algorithm can be used as a pre-processing step in a static information system or as a post-processing step on the specific documents retrieved by a query. As a pre-processor, this system assists users with deciding how to browse a database of free text documents by highlighting relevant topics and irrelevant subtopics. Such clustered data is useful for narrowing down the database over which detailed queries can be formulated. As a post-processor, this system classifies the retrieved data into clusters that capture topic categories and subcategories. The on-line algorithm can be used for constructing self-organizing information systems, for routing problems, for topic detection, and for topic tracking.

[1] In [SJJ70] stars were also identified to be potentially useful for clustering.

2 Off-line Information Organization

In this section, we begin by presenting an efficient algorithm for off-line organization of information. We then describe our system built around this algorithm, including user interface design and visualization techniques. Finally, we present a performance evaluation of our organization algorithm. We begin by examining the organization problem and introducing the star algorithm.

2.1 The Star Algorithm

We formalize our problem by representing an information system by its *similarity graph*. A similarity graph is an undirected, weighted graph $G = (V, E, w)$ where vertices in the graph correspond to documents and each weighted edge in the graph corresponds to a measure of similarity between two documents. We measure the similarity between two documents by using the cosine metric in the vector space model of the Smart information retrieval system [Sal89, Sal91].

G is a complete graph with edges of varying weight. An organization of the graph that produces reliable clusters of similarity σ (*i.e.*, clusters where documents have pairwise similarities of at least σ) can be obtained by first thresholding the graph at σ and then performing a *minimum clique cover* with maximal cliques on the resulting graph G_σ. The *thresholded graph* G_σ is an undirected graph obtained from G by eliminating every edge whose weight is lower that σ. The minimum clique cover has two features. First, by using cliques to cover the similarity graph, we are guaranteed that all the documents in a cluster have the desired degree of similarity. Second, minimal clique covers with maximal cliques allow vertices to belong to *several* clusters. In our information retrieval application this is a desirable feature as documents can have multiple subthemes. However, the algorithm can also be used to compute non-overlapping clusters. In our experimental evaluations (see Figure 4) we show that the difference in results between star with overlapping clusters and star without overlapping clusters is very small.

Unfortunately, this approach is not tractable computationally. For real corpora, similarity graphs can be very large. The clique cover problem is NP-complete, and it does not admit polynomial-time approximation algorithms [LY94, Zuc93]. While we cannot perform a clique cover nor even approximate such a cover, we can instead cover our graph by *dense subgraphs*. What we lose in intra-cluster similarity guarantees, we gain in computational efficiency. In this section and the sections that follow, we describe off-line and on-line covering algorithms and analyze their performance and efficiency.

We approximate a clique cover by covering the associated thresholded similarity graph with *star-shaped subgraphs*. A star-shaped subgraph on $m+1$ vertices consists of a single *star center* and m *satellite vertices*, where there exist edges between the star center and each of the satellite vertices. While finding cliques in the thresholded similarity graph G_σ *guarantees* a pairwise similarity between documents of at least σ, it would appear at first glance that finding star-shaped subgraphs in G_σ would provide similarity guarantees between the star center and each of the satellite vertices, but no such similarity guarantees *between*

For any threshold σ:

1. Let $G_\sigma = (V, E_\sigma)$ where $E_\sigma = \{e : w(e) \geq \sigma\}$.
2. Let each vertex in G_σ initially be *unmarked.*
3. Calculate the degree of each vertex $v \in V$.
4. Let the highest degree unmarked vertex be a star center and construct a cluster from the star center and its associated satellite vertices. Mark each node in the newly constructed cluster.
5. Repeat step 4 until all nodes are marked.
6. Represent each cluster by the document corresponding to its associated star center.

Figure 1: The star algorithm

satellite vertices. However, by investigating the geometry of our problem in the vector space model, we can derive a ***lower bound*** on the similarity between satellite vertices as well as provide a formula for the *expected* similarity between satellite vertices. The latter formula predicts that the pairwise similarity between satellite vertices in a star-shaped subgraph is high, and together with empirical evidence supporting this formula, we shall conclude that covering G_σ with star-shaped subgraphs is a reliable method for clustering a set of documents.

The star algorithm is based on a greedy cover of the thresholded similarity graph by star-shaped subgraphs; the algorithm itself is summarized in Figure 1. The star algorithm is very efficient. In [APR98] we show that the star algorithm can be correctly implemented in such a way that given a thresholded similarity graph G_σ, the running time of the algorithm is $\Theta(V + E_\sigma)$, linear in the size of the input graph.

2.2 Cluster Quality

In this section, we argue that the clusters produced by a star cover have high average intra-cluster similarity weights; thus, the clusters produced are accurate and of high quality. Consider three documents C, S_1 and S_2 which are vertices in a star-shaped subgraph of G_σ, where S_1 and S_2 are satellite vertices and C is the star center. By the definition of a star-shaped subgraph of G_σ, we must have that the similarity between C and S_1 is at least σ and that the similarity between C and S_2 is also at least σ. In the vector space model, these similarities are obtained by taking the cosine of the angle between the vectors associated with each document. Let α_1 be the angle between C and S_1, and let α_2 be the angle between C and S_2. We then have that $\cos\alpha_1 \geq \sigma$ and $\cos\alpha_2 \geq \sigma$. Note that the angle between S_1 and S_2 can be at most $\alpha_1 + \alpha_2$, and therefore we have the following lower bound on the similarity between satellite vertices in a star-shaped subgraph of G_σ.

Theorem 1 *Let G_σ be a similarity graph and let S_1 and S_2 be two satellites in the same star in G_σ. If $\alpha_1 \geq \sigma$ and $\alpha_2 \geq \sigma$ are the respective similarities between S_1 and the star center and between S_2 and the star center, then the similarity between S_1 and S_2 must be at least*

$$\cos(\alpha_1 + \alpha_2) = \cos\alpha_1 \cos\alpha_2 - \sin\alpha_1 \sin\alpha_2.$$

If $\sigma = 0.7$, $\cos\alpha_1 = 0.75$ and $\cos\alpha_2 = 0.85$, for instance, we can conclude that the similarity between the two satellite vertices must be at least[2]

$$(0.75) \cdot (0.85) - \sqrt{1-(0.75)^2}\sqrt{1-(0.85)^2} \approx 0.29.$$

While this may not seem very encouraging, the above analysis is based on absolute worst-case assumptions, and in practice, the similarities between satellite vertices are much higher. We further undertook a study to determine the *expected* similarity between two satellite vertices. Under the assumption that "similar" documents are essentially "random" perturbations of one another in an appropriate vector space, we have proven the following [APR97]:

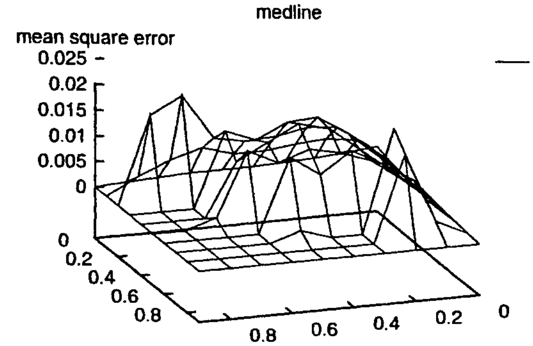


Figure 2: This figure shows the actual mean-squared prediction error for a 6,000 abstract subset of MEDLINE.

Theorem 2 *Let G_σ be a similarity graph and let S_1 and S_2 be two satellites in the same star in G_σ. If $\alpha_1 \geq \sigma$ and $\alpha_2 \geq \sigma$ are the respective similarities between S_1 and the star center and between S_2 and the star center, then the expected similarity between S_1 and S_2 is*

$$\cos\alpha_1 \cos\alpha_2 + \frac{\sigma}{1+\sigma} \sin\alpha_1 \sin\alpha_2.$$

For the previous example, the above formula would predict a similarity between satellite vertices of approximately 0.78. We have tested this formula against real data, and the results of the test with the MEDLINE data set are shown in Figure 2. In this plot, the x- and y-axes are similarities between a cluster center and each of two satellite vertices, and the z-axis is the actual mean squared prediction error of the above formula for the similarity between satellite vertices. Note that the root mean square error (RMS) is quite small (approximately 0.13 in the worst case), and for reasonably high similarities, the error is negligible. From our tests with real data, we have concluded that this formula is quite accurate and that star-shaped subgraphs are reasonably "dense" in the sense that they imply relatively high pairwise similarities between documents.

[2] Note that $\sin\theta = \sqrt{1-\cos^2\theta}$.

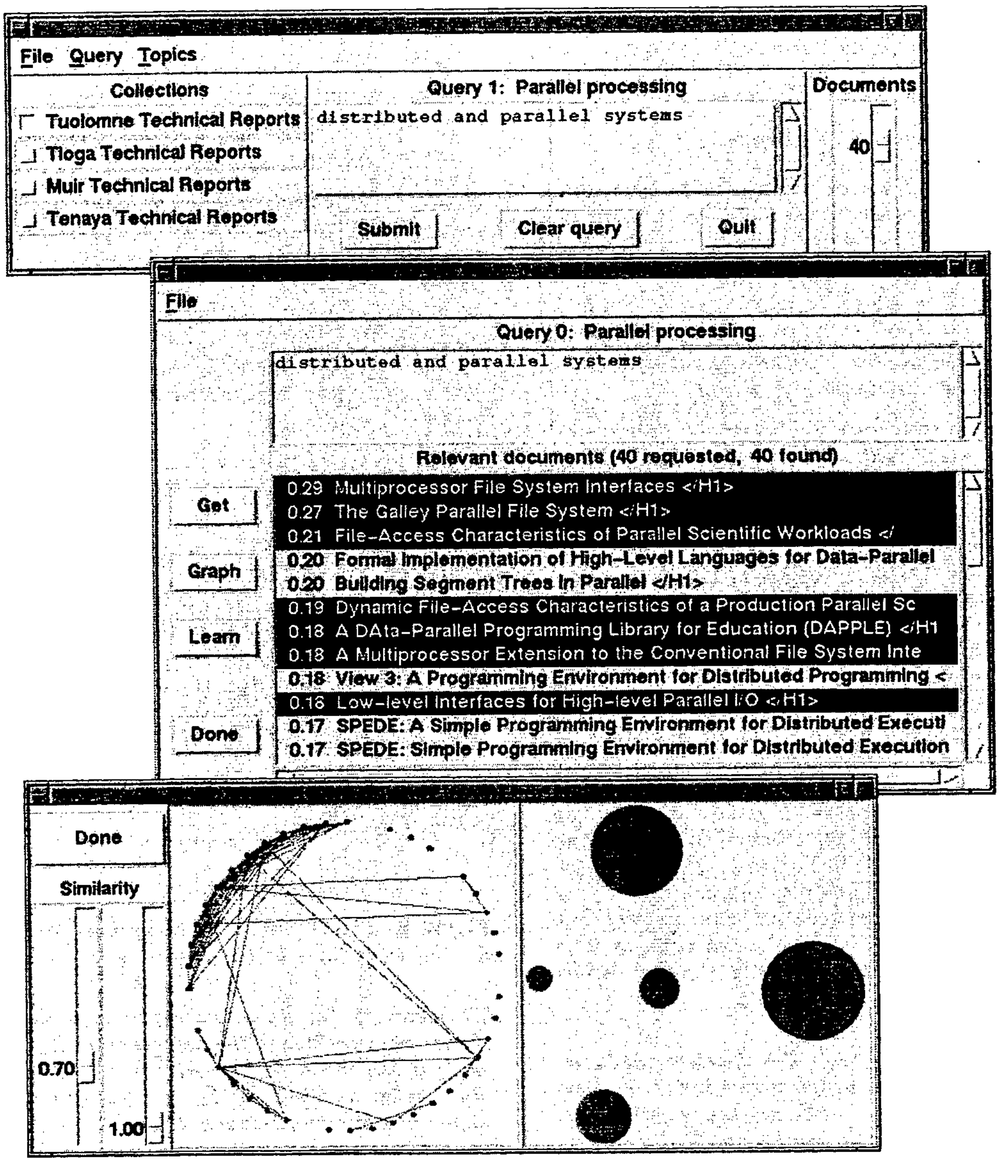


Figure 3: This is a screen snapshot from a clustering experiment. The top window is the query window. The middle window consists of a ranked list of documents that were retrieved in response to the user query. The user may select "get" to fetch a document or "graph" to request a graphical visualization of the clusters as in the bottom window. The left graph displays all the documents as dots around a circle. Clusters are separated by gaps. The edges denote pairs of documents whose similarity falls between the slider parameters. The right graph displays all the clusters as disks. The radius of a disk is proportional to the size of the cluster. The distance between the disks is proportional to the similarity distance between the clusters.

2.3 The System

We have implemented a system for organizing information that uses the star algorithm. This organization system was used for the experiments described in this paper. It consists of an augmented version of the Smart system [Sal91, All95], a user interface we have designed, and an implementation of the star algorithm on top of Smart. To index the documents we used the Smart search engine with a cosine normalization weighting scheme. We enhanced Smart to compute a document to document similarity matrix for a set of retrieved documents or a whole collection. The similarity matrix is used to compute clusters and to visualize the clusters. The user interface is implemented in Tcl/Tk.

The organization system can be run on a whole collection, on a specified subcollection, or on the collection of

documents retrieved in response to a user query. Users can input queries by typing free text. They have the choice of specifying several corpora. This system supports distributed information retrieval, but in this paper we do not focus on distribution and we assume only one centrally located corpus. In response to a user query, Smart is invoked to produce a ranked list of the top most relevant documents, their titles, locations and document-to-document similarity information. The similarity information for the entire collection, or for the collection computed by the query engine is provided as input to the star algorithm. This algorithm returns a list of clusters and marks their centers.

2.4 Visualization

We have developed a visualization method for organized data that presents users with three views of the data (see Figure 3): a list of text titles, a graph that shows the similarity relationship between the documents, and a graph that shows the similarity relationship between the clusters. These views provide users with summaries of the data at different levels of detail (text, document and topic) and facilitate browsing by topic structure.

The connected graph view (inspired by [All95]) has nodes corresponding to the retrieved documents. The nodes are placed in a circle, with nodes corresponding to the same cluster placed together. Gaps between the nodes allow us to identify clusters easily. Edges between nodes are color coded according to the similarity between the documents. Two slider bars allow the user to establish minimal and maximal weight of edges to be shown.

Another view presents clusters as disks whose size is proportional to the size of the corresponding cluster. The distance between two clusters is defined as a distance between the central documents and captures the topic separation between the clusters. Simulated annealing is used to find a cluster placement that minimizes the sum of relative distance errors between clusters. We selected a cooling schedule $\alpha(t) = t/(1+\beta t)$, where $\beta = 10^{-3}$, initial temperature is 500 and the freezing point is 10^{-2}. This setting provides a good placement when the number of clusters returned by the algorithm is small. This algorithm is fast, and its running time does not depend on the number of clusters. When the number of clusters is large, the ellipsoid-based method for Euclidean graph embeddings described in [LLR95] can be used instead.

All three views and a title window allow the user to select an individual document or a cluster. A selection made in one window is simultaneously reflected in the others.

2.5 Performance Comparison with Two Clustering Algorithms

In order to evaluate the performance of our system, we tested the star algorithm against two classic clustering algorithms: the single link method [Cro77] and the average link method [Voo85]. We used data from the TREC-6 conference as our testing medium. The TREC collection contains a set of 130,471 documents of which 21,694 have been ascribed relevance data with respect to 47 topics. These 21,694 documents were partitioned into 22 separate subcollections of approximately 1,000 documents each. Within a subcollection, each of the 47 topics has a corresponding subset of documents which is relevant to that topic.

The goal of a clustering method is to organize the set of documents in such a way that the subset of documents corresponding to a selected topic appears as a cluster in the organization. For each of the subcollections, we performed the following experiment. Given a selected topic, the set of documents was organized by a clustering method in question, and the "best" cluster corresponding to this topic was returned. Two issues immediately arise: first, how does one measure the "quality" of a cluster to determine which is "best"; and second, how does one appropriately generate clusters from which to choose. To measure the quality of a cluster, we use the common E measure [Rij79] as defined below

$$E(p,r) = 1 - \frac{2}{1/p + 1/r}$$

where p and r are the standard *precision* and *recall* of the cluster with respect to the set of documents relevant to the topic. Note that $E(p,r)$ is simply one minus the harmonic mean of the precision and recall; thus, $E(p,r)$ ranges from 0 to 1 where $E(p,r) = 0$ corresponds to perfect precision and recall and $E(p,r) = 1$ corresponds to zero precision and recall. It is worthwhile to note that when viewing data comparing two clustering methods, *lower* $E(p,r)$ values correspond to *better* performance. In order to compare the clustering methods fairly, each of the methods was run in such a way so as to produce the "best" possible cluster with respect to a given topic, as defined by the $E(p,r)$ measure above. (This is in keeping with previous comparative analyses of clustering methods; see, for example, Burgin [Bur95] and Shaw [Sha93].) In the case of the single link and star cover algorithms, the algorithms were run using a range of thresholds, and the best cluster obtained over all thresholds was returned. (One can view the clustering obtained with respect to a given threshold as a "slice" within a hierarchical clustering over all thresholds; thus, in effect, the best cluster in the hierarchy was returned in these experiments.) In the case of the average-link algorithm which naturally produces a hierarchical clustering, the best cluster within the hierarchy was returned.

Unlike the star algorithm, single and average link algorithms do not allow overlapping clusters. It has been suggested [All98] that the differences in performance may be attributed to the effects of overlapping rather than to the actual properties of the algorithm. To investigate this issue we conducted the same experiments using a version of the star clustering algorithm that eliminates the overlapping clusters. In this setting we used the star algorithm to find a set of star centers, then partitioned a collection by assigning a document to the closest star center. This methodology has been used before [JD88]. We note that the difference in results between star with overlapping clusters and star without overlapping clusters is very small. Both algorithms still outperform single link and average link (See Figure 4).

Each subcollection of 1,000 documents corresponded to an individual experiment. For a given clustering method, the appropriate algorithm was employed to determine the best possible cluster (as defined by the $E(p,r)$ measure) for each of the 47 topics. For each optimal cluster, the $E(p,r)$, precision and recall values were calculated with respect to the actual set of documents relevant to the topic,

coll	star (overlap)			star (partition)			average link			single link		
	p	r	E	p	r	E	p	r	E	p	r	E
1	0.78	0.56	0.35	0.79	0.53	0.36	0.74	0.50	0.40	0.77	0.47	0.41
2	0.74	0.59	0.35	0.70	0.55	0.38	0.88	0.43	0.43	0.88	0.41	0.44
3	0.78	0.53	0.37	0.79	0.48	0.41	0.84	0.44	0.43	0.83	0.43	0.43
4	0.76	0.50	0.40	0.81	0.46	0.41	0.71	0.46	0.44	0.73	0.41	0.48
5	0.80	0.50	0.38	0.78	0.50	0.39	0.85	0.40	0.46	0.81	0.40	0.46
6	0.76	0.41	0.47	0.68	0.45	0.46	0.78	0.39	0.48	0.83	0.34	0.51
7	0.76	0.62	0.32	0.79	0.61	0.31	0.81	0.52	0.36	0.78	0.50	0.39
8	0.75	0.57	0.35	0.73	0.57	0.36	0.82	0.48	0.39	0.86	0.44	0.42
9	0.82	0.49	0.39	0.80	0.50	0.38	0.89	0.44	0.41	0.87	0.43	0.43
10	0.74	0.52	0.39	0.79	0.46	0.42	0.85	0.42	0.44	0.87	0.38	0.47
11	0.82	0.55	0.34	0.86	0.48	0.38	0.83	0.45	0.42	0.85	0.44	0.42
12	0.80	0.55	0.35	0.83	0.53	0.36	0.82	0.49	0.38	0.83	0.40	0.46
13	0.81	0.53	0.36	0.81	0.49	0.39	0.84	0.46	0.40	0.89	0.40	0.44
14	0.76	0.47	0.42	0.73	0.46	0.43	0.86	0.36	0.50	0.91	0.31	0.54
15	0.75	0.54	0.37	0.79	0.48	0.40	0.83	0.35	0.50	0.87	0.33	0.52
16	0.87	0.47	0.39	0.86	0.46	0.40	0.91	0.40	0.45	0.95	0.39	0.45
17	0.64	0.53	0.42	0.64	0.51	0.43	0.80	0.39	0.48	0.76	0.39	0.48
18	0.77	0.56	0.35	0.81	0.51	0.37	0.79	0.53	0.36	0.81	0.48	0.40
19	0.73	0.54	0.38	0.74	0.50	0.40	0.83	0.42	0.45	0.85	0.39	0.46
20	0.71	0.51	0.41	0.76	0.48	0.41	0.81	0.41	0.45	0.86	0.36	0.49
21	0.74	0.61	0.33	0.79	0.56	0.34	0.84	0.49	0.38	0.88	0.46	0.40
22	0.76	0.63	0.31	0.80	0.60	0.32	0.83	0.47	0.40	0.85	0.46	0.40
avg	0.77	0.54	0.37	0.78	0.51	0.39	0.83	0.44	0.43	0.84	0.41	0.45

Figure 4: This figure shows comparison data for the star algorithm, the partitioning star algorithm, the single link algorithm, and the average link algorithm for 22 subcollections of TREC documents. For each algorithm, p represents the average precision computed across all clusters found for the collection; r represents the average recall computed across all clusters found for the collection; and $E(p,r)$ is the aggregate measure $1 - \frac{2}{1/p+1/r}$.

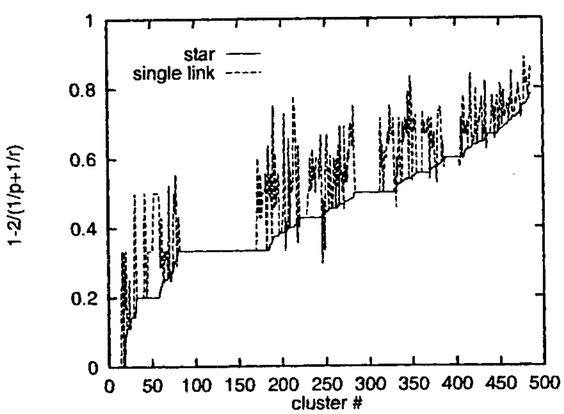


Figure 5: This figure shows the $E(p,r)$ measure for the partitioning star clustering algorithm and for the single link clustering algorithm. The y axis shows the $E(p,r)$ measure, while the x axis shows the cluster number. Clusters have been sorted according to the $E(p,r)$ values of the star algorithm.

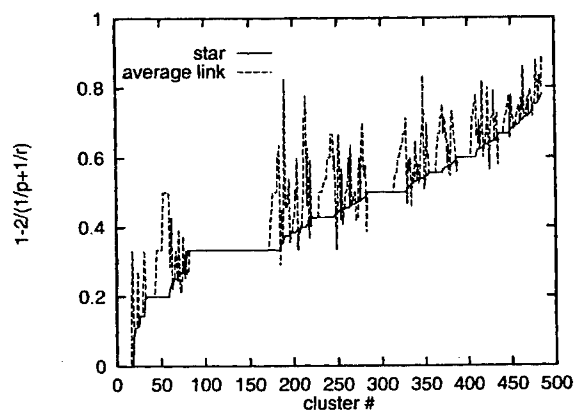


Figure 6: This figure shows the $E(p,r)$ measure for the partitioning star clustering algorithm and for the average link clustering algorithm. The y axis shows the $E(p,r)$ measure, while the x axis shows the cluster number. Clusters have been sorted according to the $E(p,r)$ values of the star algorithm.

and these values were averaged over all topics to obtain the three numbers reported for each experiment and clustering method in Figure 4. Averaging over all 22 experiments, we find that the mean $E(p,r)$ values for star, partitioning star, average link and single link are 0.37, 0.39, 0.43 and 0.45, respectively. Thus, the star algorithm represents a 16.2% improvement in performance with respect to average link and an 21.6% improvement with respect to single link. The difference is only partly due to the effect of allowing overlapping clusters - the partitioning star algorithm still gives us a 10.2% and 15.4% improvement in performance over average link and single link respectively.

We repeated this experiment on the same data, using one collection only (of 21,694 documents.) The precision, recall, and E values for star (overlap), star, average link, and single link were (.52, .36, .58), (.53, .32, .61), (.63, .25, .64), and (.66, .20, .70) respectively. We note that the E measures are worse for all four algorithms on this larger collection and that the star algorithm outperforms average link by 10.3% and single link by 20.7%.

Figures 5 and 6 show detailed $E(p,r)$ values for the star algorithm vs. the single link algorithm and for the star algorithm vs. the average link algorithm over the collection of experiments. Each cluster computed by the algorithm has

an $E(p,r)$ value. For better readability of these graphs, we sorted the clusters produced by the star algorithm according to their $E(p,r)$ values. We plotted the corresponding $E(p,r)$ values for the single link algorithm (see the oscillating line in Figure 5) and for the average link algorithm (see the oscillating line in Figure 6). We note that the $E(p,r)$ values for the star clusters are almost everywhere lower than the corresponding values for the single link and average link algorithms; thus, the star algorithm outperforms these two methods.

These experiments show that the star algorithm outperforms the single link and average link methods. Since the star algorithm is also simple to implement and highly efficient, we believe that the star algorithm is very effective for information organization and other text clustering applications.

3 On-line Information Organization

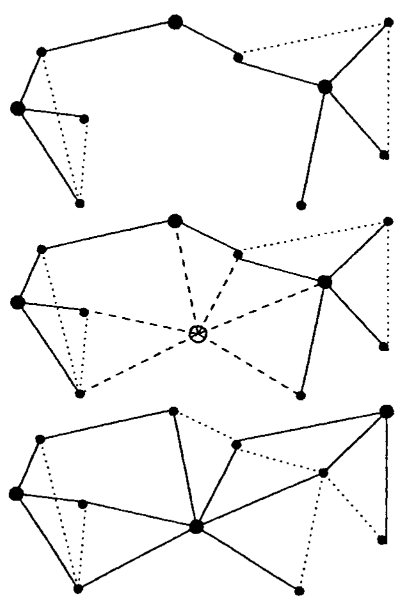


Figure 7: This figure shows the star cover change after the insertion of a new vertex. The larger-radius disks denote star centers, the other disks denote satellite vertices. The star edges are denoted by solid lines. The inter-satellite edges are denoted by dotted lines. The top figure shows an initial graph and its star cover. The middle figure shows the graph after the insertion of a new document. The bottom figure shows the star cover of the new graph.

In this section we consider algorithms for computing the organization of a dynamic information system. We derive an on-line version of the star algorithm for information organization that can incrementally compute clusters of similar documents. We continue assuming the vector space model and the cosine metric to capture the pairwise similarity between the documents of the corpus.

3.1 The On-line Star Algorithm

We assume that documents are inserted or deleted from the collection one at a time. For simplicity, we will focus our discussion on adding documents to the collection. The delete algorithm is similar. The intuition behind the incremental computation of the star cover of a graph after a new vertex is inserted is depicted in Figure 7. The top figure denotes a thresholded similarity graph and a correct star cover for this graph. Suppose a new vertex is inserted in the graph, as in the middle figure. The original star cover is no longer correct for the new graph. The bottom figure shows the correct star cover for the new graph. How does the addition of this new vertex affect the correctness of the star cover? In general, the answer depends on the degree of the new vertex and on its adjacency list. If the adjacency list of the new vertex does not contain any star centers, the new vertex can be added in the star cover as a star center. If the adjacency list of the new vertex contains any center vertex c whose degree is higher, the new vertex becomes a satellite vertex of c. The difficult case that destroys the correctness of the star cover is when the new vertex is adjacent to a collection of star centers, each of whose degree is lower than that of the new vertex. In this situation, the star structure already in place has to be modified to assign the new vertex as a star center. The satellite vertices in the stars that are broken as a result have to be re-evaluated.

Motivated by the intuition in the previous paragraph, we now describe an on-line algorithm for incrementally computing star covers of dynamic graphs. The algorithm is shown in Figure 8. This algorithm uses a special data structure to efficiently maintain the star cover of an undirected graph $G = (V, E)$. For each vertex $v \in V$, we maintain the following data.

$v.type$	satellite or center
$v.degree$	degree of v
$v.adj$	list of adjacent vertices
$v.centers$	list of adjacent centers
$v.inQ$	flag specifying if v being processed

Note that while $v.type$ can be inferred from $v.centers$ and $v.degree$ can be inferred from $v.adj$, it will be convenient to have all five pieces of data in the algorithm. Let α be a vertex to be added to G, and let L be the list of vertices in G which are adjacent to α. The algorithm in Figure 8 will appropriately update the star cover of G. See [APR97] for a more detailed correctness argument.

3.2 Analysis

We have shown that the star cover produced by the on-line star algorithm is correct in that it is identical to the star cover produced by the off-line algorithm (or one of the correct covers, if more than one exists) [APR97]. Furthermore, the on-line star algorithm is very efficient. In our initial tests, we have implemented the on-line star algorithm using a heap for the priority queue and simple linked lists for the various lists required. The time required to insert a new vertex and associated edges into a thresholded similarity graph and to appropriately update the star cover is largely governed by the number of stars that are broken during the update, since breaking stars requires inserting new elements into the priority queue. In practice, very few stars are broken during any given update (see Figure 9). This is due partly to the fact that relatively few stars exist at any given time (as compared to the number of vertices or edges in the thresholded similarity graph) and partly to

the fact that the likelihood of breaking any individual star is also small [APR97].

```
UPDATE(α, L)
  α.type ← satellite
  α.degree ← 0
  α.adj ← ∅
  α.centers ← ∅
  forall β in L
    α.degree ← α.degree + 1
    β.degree ← β.degree + 1
    INSERT(β, α.adj)
    INSERT(α, β.adj)
    if (β.type = center)
      INSERT(β, α.centers)
    else
      β.inQ ← true
      ENQUEUE(β, Q)
    endif
  endfor
  α.inQ ← true
  ENQUEUE(α, Q)
  while (Q ≠ ∅)
    φ ← EXTRACTMAX(Q)
    if (φ.centers = ∅)
      φ.type ← center
      forall β in φ.adj
        INSERT(φ, β.centers)
      endfor
    else
      if (∀δ ∈ φ.centers, δ.degree < φ.degree)
        φ.type ← center
        forall β in φ.adj
          INSERT(φ, β.centers)
        endfor
        forall δ in φ.centers
          δ.type ← satellite
          forall μ in δ.adj
            DELETE(δ, μ.centers)
            if (μ.inQ = false)
              μ.inQ ← true
              ENQUEUE(μ, Q)
            endif
          endfor
        endfor
      endif
    endif
    φ.inQ ← false
  endwhile
```

Figure 8: The on-line star algorithm for clustering.

We evaluated the on-line star cover algorithm on a 2224 document corpus consisting of a judged subcollection of TREC documents augmented with our department's technical reports. We ran 4 experiments. Each time we selected a different threshold and proceeded to insert the 2224 documents in random order, using the on-line star cluster algorithm. The results of these experiments were averaged. The running time measurements appear to be linear in the number of edges of the similarity graph. Figures 9 and 10 show the experimental data. Note that the number of broken stars is roughly linear in the number of vertices, the running time is linear in the number of edges in the graph, although we can see the effects of lower order terms.

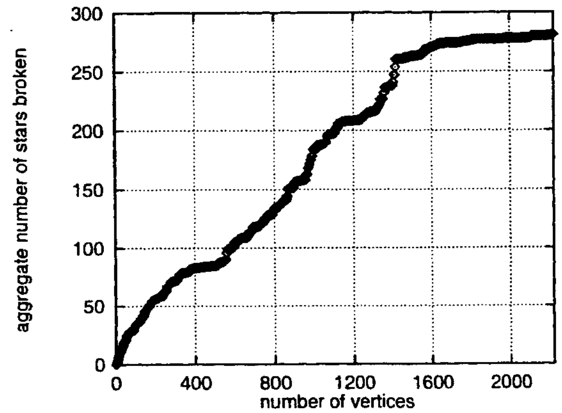


Figure 9: The dependence of number of broken stars on the number of vertices for TREC data.

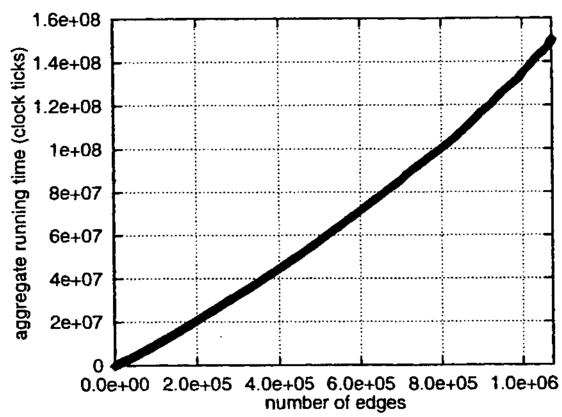


Figure 10: This figure shows the dependence of the running time of the on-line star algorithm on the number of edges in a TREC subcollection.

4 Discussion

We have presented, analyzed, and evaluated the star clustering algorithm for information organization. We described an off-line version of this algorithm that can be used to organize static information in accurate clusters efficiently. We also described an on-line version of the algorithm that can be used to organize dynamic data for tasks that require incremental updates in the topic structure of the corpus, such as the routing task, the new topic detection task, and the topic tracking task.

Our implementation of this algorithm contributes a novel visualization method for clusters that presents users with

disks whose radii correspond to the cluster size and that are embedded in the plane so as to capture the topic distance between the clusters.

We evaluated the star algorithm by comparing it against the single link and the average link algorithms in several experiments with TREC data. We found that the star algorithm outperforms the single link algorithm and the average link algorithm. Since the star algorithm is faster and easier to implement and than the average link algorithm, we advocate its use. The on-line algorithm produces the same clustering as the off-line algorithm. Thus, our evaluation of the off-line star algorithm also suggests using the on-line star algorithm for tasks that require computing the topic structure incrementally and adaptively.

Our findings so far suggest using the star algorithm for a variety of tasks. We are currently conducting experiments using the on-line star algorithm for new topic detection and topic tracking. Because of its cluster quality, efficiency, and incremental properties, we believe this algorithm will lead to improved results in solving these tasks.

Acknowledgements

We thank James Allan for many useful comments and suggestions on this research. This work is supported in part by ONR contract N00014-95-1-1204, Rome Labs contract F30602-98-C-0006, and Air Force MURI contract F49620-97-1-0382. We are grateful for this support.

References

[AB84] M. Aldenderfer and R. Blashfield, *Cluster Analysis*, Sage, Beverly Hills, 1984.

[All95] J. Allan. *Automatic hypertext construction.* PhD thesis. Department of Computer Science, Cornell University, January 1995.

[All98] J. Allan, Personal communication, March 1998.

[APR98] J. Aslam, K. Pelekhov, and D. Rus, Generating, visualizing, and evaluating high-accuracy clusters for information organization, in *Principles of Digital Document Processing*, eds. C. Nicholas, Lecture Notes in Computer Science, Sprinter Verlag 1998 (to appear). Also available as Technical Report PCS-TR97-319, Department of Computer Science, Dartmouth, 1997.

[APR97] J. Aslam, K. Pelekhov, and D. Rus, Computing Dense Clusters On-line for Information Organization, Technical Report PCS-TR97-324, Department of Computer Science, Dartmouth, 1997.

[Bol95] B. Bollobás, *Random Graphs*, Academic Press, London, 1995.

[Bur95] R. Burgin, The retrieval effectiveness of five clustering algorithms as a function of indexing exhaustively, *Journal of American Society for Information Science* 46(8:562-572, 1995.

[Can93] F. Can, Incremental clustering for dynamic information processing, in *ACM Transactions on Information Systems*, no. 11, pp143-164, 1993.

[CCFM97] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, Incremental clustering and dynamic information retrieval, in *Proceedings of the* 29^{th} *Symposium on Theory of Computing*, 1997.

[Cro80] W. B. Croft. A model of cluster searching based on classification. *Information Systems*, 5:189-195, 1980.

[Cro77] W. B. Croft. Clustering large files of documents using the single-link method. *Journal of the American Society for Information Science*, pp189-195, November 1977.

[CKP93] D. Cutting, D. Karger, and J. Pedersen. Constant interaction-time Scatter/Gather browsing of very large document collections. In *Proceedings of the* 16^{th} *SIGIR*, 1993.

[FG88] T. Feder and D. Greene, Optimal algorithms for approximate clustering, in *Proceedings of the* 20^{th} *Symposium on Theory of Computing*, pp 434-444, 1988.

[HP96] M. Hearst and J. Pedersen. Reexamining the cluster hypothesis: Scatter/Gather on Retrieval Results. In *Proceedings of the* 19^{th} *SIGIR*, 1996.

[HS86] D. Hochbaum and D. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *Journal of the ACM*, no. 33, pp533-550, 1986.

[JD88] A. Jain and R. Dubes. *Algorithms for Clustering Data*, Prentice Hall 1988.

[JR71] N. Jardine and C.J. van Rijsbergen. The use of hierarchical clustering in information retrieval, *Information Storage and Retrieval*, 7:217-240, 1971.

[KP93] G. Kortsarz and D. Peleg. On choosing a dense subgraph. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS)*, 1993.

[LLR95] N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica* 15(2):215-245, 1995.

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM* 41, 960–981, 1994.

[Rij79] C.J. van Rijsbergen. *Information Retrieval.* Butterworths, London, 1979.

[Sal89] G. Salton. *Automatic Text Processing: the transformation, analysis, and retrieval of information by computer*, Addison-Wesley, 1989.

[Sal91] G. Salton. The Smart document retrieval project. In *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pp 356-358.

[Sha86] W. Shaw, On the foundation of evaluation, *Journal of the American Society for Information Science*, vol 37, pp 346-348, 1986.

[Sha93] W. Shaw, Controlled and uncontrolled subject descriptions in the CF database: a comparison of optimal cluster-based retrieval results, *Information Processing and Management*, vol. 29, pp 751-763, 1993.

[SJJ70] K. Spark Jones and D. Jackson. The use of automatically-obtained keyword classifications for information retrieval. *Information Storage and Retrieval*, 5:174-201, 1970.

[Tur90] H. Turtle. Inference networks for document retrieval. PhD thesis. University of Massachusetts, Amherst, 1990.

[vRC75] C.J. van Rijsbergen and B. Croft, Document clustering: an evaluation of some experiments with the Cranfield 1400 collection, *Information Processing and Management*, 11, 171-182.

[Voo85] E. Voorhees. The effectiveness and efficiency of agglomerative hierarchical clustering in document retrieval, PhD Thesis, Department of Computer Science, Cornell University 1985, available as TR 85-705.

[Voo85] E. Voorhees. The cluster hypothesis revisited. In *Proceedings of the* 8^{th} *SIGIR*, pp 95-104, 1985.

[Wil88] P. Willett. Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management*, 24:(5):577-597, 1988.

[Wor71] S. Worona. Query clustering in a large document space. In Ed. G. Salton, *The SMART Retrieval System*, pp 298-310. Prentice-Hall, 1971.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *Proceedings of the Eight Annual Structure in Complexity Theory Conference*, IEEE Computer Society, 305–312, 1993.

A Practical Clustering Algorithm for Static and Dynamic Information Organization*

Javed Aslam Katya Pelekhov Daniela Rus
Dartmouth College†

Abstract

We present and analyze the off-line star algorithm for clustering static information systems and the on-line star algorithm for clustering dynamic information systems. These algorithms organize a document collection into a number of clusters that is naturally induced by the collection via a computationally efficient cover by dense subgraphs. We further show a lower bound on the accuracy of the clusters produced by these algorithms as well as demonstrate that these algorithms are efficient (running times roughly linear in the size of the problem). Finally, we provide data from a number of experiments.

1 Introduction

We wish to create more versatile information capture and access systems for digital libraries by using information organization: thousands of electronic documents will be organized automatically as a hierarchy of topics and subtopics, using algorithms grounded in geometry, probabilities, and statistics. Off-line information organization algorithms will be useful for organizing static collections (for example, large-scale legacy data). Incremental, on-line information organization algorithms will be useful to keep dynamic corpora, such as news feeds, organized. Current information systems such as Inquery [Tur90], Smart [Sal91], or Alta Vista provide some simple automation by computing ranked (sorted) lists of documents, but it is ineffective for users to scan a list of hundreds of document titles. To cull the relevant information out of a large set of potentially useful dynamic sources we need methods for organizing and reorganizing dynamic information as accurate clusters, and ways of presenting users with the topic summaries at various levels of detail.

There has been extensive research on clustering and applications to many domains [HS86, AB84]. For a good overview see [JD88]. For a good overview of using clustering in Information Retrieval (IR) see [Wil88]. The use of clustering in IR was mostly driven by *the cluster hypothesis* [Rij79] which states that relevant documents tend to be more closely related to each other than to non-relevant documents. Jardine and van Rijsbergen [JR71] show some evidence that search results could be improved by clustering. Hearst and Pedersen [HP96] re-examine the cluster hypothesis by focusing on the Scatter/Gather system [CKP93] and conclude that it holds for browsing tasks.

Systems like Scatter/Gather [CKP93] provide a mechanism for user-driven organization of data in a fixed number of clusters, but the users need to be in the loop and the computed clusters do not have accuracy guarantees. Scatter/Gather uses fractionation to compute nearest-neighbor clusters. Charika, et al. [CCFM97] consider a dynamic clustering algorithm to partition a collection of text documents into a *fixed* number of clusters. Since in dynamic information systems the number of topics is not known *a priori*, a fixed number of clusters cannot generate a natural partition of the information.

Our work on clustering presented in this paper and in [APR97] provides positive evidence for the cluster hypothesis. We propose an off-line algorithm for clustering static information and an on-line version of this algorithm for clustering dynamic information. These two algorithms compute clusters induced by the natural topic structure of the space. Thus, this work is different than [CKP93, CCFM97] in that we do not impose the constraint to use a fixed number of clusters. As a result, we can guarantee a lower bound on the topic similarity between the documents in each cluster. The model for topic similarity is the standard vector space model used in the information retrieval community [Sal89] which is explained in more detail in this paper in Section 2.

To compute accurate clusters, we formalize clustering as covering graphs by cliques. Covering by cliques is NP-complete, and thus intractable for large document collections. Unfortunately, it has also been shown that the problem cannot even be approximated in polynomial

*Research partially supported by ONR contract N00014-95-1-1204, Rome Labs contract F30602-98-C-0006, and Air Force MURI contract F49620-97-1-0382.

†Email: {jaa,katya,rus}@cs.dartmouth.edu

time [LY94, Zuc93]. We instead use a cover by *dense subgraphs* that are *star-shaped* and that can be computed *off-line* for static data and *on-line* for dynamic data. We show that the off-line and on-line algorithms produce correct clusters efficiently. Asymptotically, the running time of both algorithms is roughly linear in the size of the similarity graph that defines the information space (explained in detail in Section 2). We also show lower bounds on the topic similarity within the computed clusters (a measure of the accuracy of our clustering algorithm) as well as provide experimental data.

Finally, we compare the performance of the star algorithm to two widely used algorithms for clustering in IR and other settings: the single link method[1] [Cro77] and the average link algorithm[2] [Voo85]. Neither algorithm provides guarantees for the topic similarity within a cluster. The single link algorithm can be used in off-line and on-line mode, and it is faster than the average link algorithm, but it produces poorer clusters than the average link algorithm. The average link algorithm can only be used off-line to process static data. The star clustering algorithm, on the other hand, computes topic clusters that are naturally induced by the collection, provides guarantees on cluster quality, computes more accurate clusters than either the single link or average link methods, is efficient, admits an efficient and simple on-line version, and can perform hierarchical data organization. We describe experiments in this paper with the TREC[3] database demonstrating these abilities.

Our algorithms for organizing information systems can be used in several ways. The off-line algorithm can be used as a pre-processing step in a static information system or as a post-processing step on the specific documents retrieved by a query. As a pre-processor, this system assists users with deciding how to browse a database of free text documents by highlighting relevant topics and irrelevant subtopics. Such clustered data is useful for narrowing down the database over which detailed queries can be formulated. As a post-processor, this system classifies the retrieved data into clusters that capture topic categories and subcategories. The on-line algorithm can be used as a basis for constructing self-organizing information systems. As the content of dynamic information systems changes, the on-line algorithm can efficiently automate the process of organization and re-organization to compute accurate topic summaries at various level of similarity.

[1] In the single link clustering algorithm a document is part of a cluster if it is "related" to at least *one* document in the cluster.

[2] In the average link clustering algorithm a document is part of a cluster if it is "related" to an average number of documents in the cluster.

[3] Text Retrieval Conference

2 Clustering static data with star-shaped subgraphs

In this section we motivate and present an off-line algorithm for organizing information systems. The algorithm is very simple and efficient, and it computes high-density clusters.

We formulate our problem by representing an information system by its *similarity graph*. A similarity graph is an undirected, weighted graph $G = (V, E, w)$ where vertices in the graph correspond to documents and each weighted edge in the graph corresponds to a measure of similarity between two documents. We measure the similarity between two documents by using the cosine metric in the vector space model of the Smart information retrieval system [Sal91, Sal89].

The vector space model for textual information aggregates statistics on the occurrence of words in documents. The premise of the vector space model is that two documents are similar if they use the same words. A vector space can be created for a collection (or corpus) of documents by associating each important word in the corpus with one dimension in the space. The result is a high dimensional vector space. Documents are mapped as points in this space according to their word frequencies. Similar documents map to nearby points. In the vector space model, document similarity is measured by the angle between the corresponding document vectors. The standard in the information retrieval community is to map the angles to the interval $[0, 1]$ by taking the cosine of the vector angles.

G is a complete graph with edges of varying weight. An organization of the graph that produces reliable clusters of similarity σ (*i.e.*, clusters where documents have pairwise similarities of at least σ) can be obtained by (1) thresholding the graph at σ and (2) performing a *minimum clique cover* with maximal cliques on the resulting graph G_σ. The *thresholded graph* G_σ is an undirected graph obtained from G by eliminating all the edges whose weights are lower that σ. The minimum clique cover has two features. First, by using cliques to cover the similarity graph, we are guaranteed that all the documents in a cluster have the desired degree of similarity. Second, minimal clique covers with maximal cliques allow vertices to belong to *several* clusters. In our information retrieval application this is a desirable feature as documents can have multiple subthemes.

Unfortunately, this approach is computationally intractable. For real corpora, similarity graphs can be very large. The clique cover problem is NP-complete, and it does not admit polynomial-time approximation algorithms [LY94, Zuc93]. While we cannot perform

a clique cover nor even approximate such a cover, we can instead cover our graph by *dense subgraphs*. What we lose in intra-cluster similarity guarantees, we gain in computational efficiency. In the sections that follow, we describe off-line and on-line covering algorithms and analyze their performance and efficiency.

2.1 Dense Star-Shaped Covers

We approximate a clique cover by covering the associated thresholded similarity graph with *star-shaped subgraphs*. A star-shaped subgraph on $m+1$ vertices consists of a single *star center* and m *satellite vertices*, where there exist edges between the star center and each of the satellite vertices. While finding cliques in the thresholded similarity graph G_σ *guarantees* a pairwise similarity between documents of at least σ, it would appear at first glance that finding star-shaped subgraphs in G_σ would provide similarity guarantees between the star center and each of the satellite vertices, but no such similarity guarantees *between satellite vertices*. However, by investigating the geometry of our problem in the vector space model, we can derive a *lower bound* on the similarity between satellite vertices as well as provide a formula for the *expected* similarity between satellite vertices. The latter formula predicts that the pairwise similarity between satellite vertices in a star-shaped subgraph is high, and together with empirical evidence supporting this formula, we shall conclude that covering G_σ with star-shaped subgraphs is an accurate method for clustering a set of documents.

Consider three documents C, S_1 and S_2 which are vertices in a star-shaped subgraph of G_σ, where S_1 and S_2 are satellite vertices and C is the star center. By the definition of a star-shaped subgraph of G_σ, we must have that the similarity between C and S_1 is at least σ and that the similarity between C and S_2 is also at least σ. In the vector space model, these similarities are obtained by taking the cosine of the angle between the vectors associated with each document. Let α_1 be the angle between C and S_1, and let α_2 be the angle between C and S_2. We then have that $\cos\alpha_1 \geq \sigma$ and $\cos\alpha_2 \geq \sigma$. Note that the angle between S_1 and S_2 can be at most $\alpha_1 + \alpha_2$, and therefore we have the following lower bound on the similarity between satellite vertices in a star-shaped subgraph of G_σ.

PROPOSITION 2.1. *Let G_σ be a similarity graph and let S_1 and S_2 be two satellites in the same star in G_σ. Then the similarity between S_1 and S_2 must be at least*

$$\cos(\alpha_1+\alpha_2) = \cos\alpha_1\cos\alpha_2 - \sin\alpha_1\sin\alpha_2.$$

If $\sigma = 0.7$, $\cos\alpha_1 = 0.75$ and $\cos\alpha_2 = 0.85$, for instance, we can conclude that the similarity between the two satellite vertices must be at least[4]

$$(0.75)\cdot(0.85) - \sqrt{1-(0.75)^2}\sqrt{1-(0.85)^2} \approx 0.29.$$

Note that while this may not seem very encouraging, the above analysis is based on absolute worst-case assumptions, and in practice, the similarities between satellite vertices are much higher. We further undertook a study to determine the *expected* similarity between two satellite vertices.

2.2 The random graph model

The model we use for analysis is the *random graph model* [Bol95]. A random graph $G_{n,p}$ is an undirected graph with n vertices, where each of its possible edges is inserted randomly and independently with probability p. Our problem fits the random graph model if we make the mathematical assumption that "similar" documents are essentially "random perturbations" of one another in the vector space model. This assumption is equivalent to viewing the similarity between two related documents as a random variable. By thresholding the edges of the similarity graph at a fixed value, for each edge of the graph there is a random chance (depending on whether the value of the corresponding random variable is above or below the threshold value) that the edge remains in the graph. This thresholded similarity graph is thus a random graph. While random graphs do not perfectly model the thresholded similarity graphs obtained from actual document corpora (the actual similarity graphs must satisfy various geometric constraints and will be aggregates of many "sets" of "similar" documents), random graphs are easier to analyze, and our experiments provide evidence that results obtained for random graphs closely match those obtained for thresholded similarity graphs obtained from actual document corpora. As such, we will use the random graph model for analysis and for experimental verification of the algorithms presented in this paper (in addition to experiments on actual corpora).

The following upper bound on the expected similarity between two satellite vertices holds:

PROPOSITION 2.2. *The* expected *similarity between two satellite vertices S_1 and S_2 in the same star in a similarity graph G_σ is:*

$$\cos\alpha_1\cos\alpha_2 + \frac{\sigma}{1+\sigma}\sin\alpha_1\sin\alpha_2.$$

Proof. (Omitted for space considerations.)

Note that for the previous example, the above formula would predict a similarity between satellite

[4] Note that $\sin\theta = \sqrt{1-\cos^2\theta}$.

vertices of approximately 0.78. We have tested this formula against real data, and the results of the test with the MEDLINE data set[5] are shown in Figure 1. In this plot, the x- and y-axes are similarities between cluster centers and satellite vertices, and the z-axis is the actual mean squared prediction error (MSE) of the above formula for the similarity between satellite vertices. Note that the maximum root mean squared error (RMS) is quite small (approximately 0.13 in the worst case), and for reasonably high similarities, the error is negligible. From our tests with real data, we have concluded that the random graph model holds and that this formula is quite accurate. We can conclude that star-shaped subgraphs are reasonably "dense" in the sense that they imply relatively high pairwise similarities between documents.

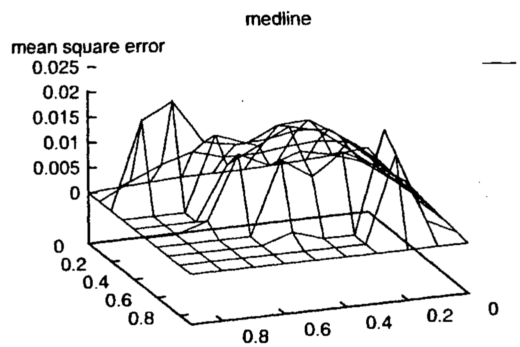


Figure 1: This figure shows the error for a 6000 abstract subset of MEDLINE.

3 The off-line star algorithm

Motivated by the discussion of the previous section, we now present the *star algorithm* which can be used to organize documents in an information system. The star algorithm is based on a greedy cover of the thresholded similarity graph by star-shaped subgraphs; the algorithm itself is summarized in Figure 2 below.

For any threshold σ:

1. Let $G_\sigma = (V, E_\sigma)$ where $E_\sigma = \{e : w(e) \geq \sigma\}$.
2. Let each vertex in G_σ initially be *unmarked*.
3. Calculate the degree of each vertex $v \in V$.
4. Let the highest degree unmarked vertex be a star center, and construct a cluster from the star center and its associated satellite vertices. Mark each node in the newly constructed cluster.
5. Repeat step 4 until all nodes are marked.
6. Represent each cluster by the document corresponding to its associated star center.

Figure 2: The star algorithm

THEOREM 3.1. *The running time of the off-line star algorithm on a similarity graph G_σ is $\Theta(V + E_\sigma)$.*

Proof. The following implementation of this algorithm has a running time *linear* in the size of the graph. Each vertex v has a data structure associate with it that contains *v.degree*, the degree of the vertex, *v.adj*, the list of adjacent vertices, *v.marked*, which is a bit denoting whether the vertex belongs to a star or not, and *v.center*, which is a bit denoting whether the vertex is a star center. (Computing *v.degree* for each vertex can easily be performed in $\Theta(V + E_\sigma)$ time.) The implementation starts by sorting the vertices in V by degree ($\Theta(V)$ time since degrees are integers in the range $\{0, |V|\}$). The program then scans the sorted vertices from the highest degree to the lowest as a greedy search for star centers. Only vertices that do not belong to a star already (that is, they are unmarked) can become star centers. Upon selecting a new star center v, its *v.center* and *v.marked* bits are set and for all $w \in v.adj$, *w.marked* is set. Only one scan of V is needed to determine all the star centers. Upon termination, the star centers and only the star centers have the *center* field set. We call the set of star centers the *star cover* of the graph. Each star is fully determined by the star center, as the satellites are contained in the adjacency list of the center vertex.

This algorithm has two features of interest. The first feature is that the star cover is not unique. A similarity graph may have several different star covers because when there are several vertices of the same highest degree, the algorithm arbitrarily chooses one of them as a star center (whichever shows up first in the sorted list of vertices). The second feature of this algorithm is that it provides a simple encoding of a star cover by assigning the types "center" and "satellite" (which is the same as "not center" in our implementation) to vertices. We define a *correct star cover* as a star cover that assigns the types "center" and "satellite" in such a way that (1) a star center is not adjacent to any other star center and (2) every satellite vertex is adjacent to at least one center vertex of higher degree. It immediately follows that:

[5]MEDLINE is a large collection of medical abstracts that is often used as benchmark in information retrieval experiments.

THEOREM 3.2. *The off-line star algorithm produces a correct star cover.*

We will use the two features of the off-line algorithm mentioned above in the analysis of the on-line version of the star algorithm, in the next section.

4 Clustering dynamic data with the star algorithm

In this section we consider algorithms for computing the organization of a dynamic information system. We derive an on-line version of the star algorithm for information organization that can incrementally compute clusters of similar documents. We continue assuming the vector space model and the cosine metric to capture the pairwise similarity between the documents of the corpus, and the random graph model for analyzing the expected behavior of the new algorithm.

We assume that documents are inserted or deleted from the collection one at a time. For simplicity, we will focus our discussion on adding documents to the collection. The delete algorithm is similar. The intuition behind the incremental computation of the star cover of a graph after a new vertex is inserted is depicted in Figure 3. The top figure denotes a similarity graph and a correct star cover for this graph. Suppose a new vertex is inserted in the graph, as in the middle figure. The original star cover is no longer correct for the new graph. The bottom figure shows the correct star cover for the new graph. How does the addition of this new vertex affect the correctness of the star cover? In general, the answer depends on the degree of the new vertex and on its adjacency list. If the adjacency list of the new vertex does not contain any star centers, the new vertex can be added in the star cover as a star center. If the adjacency list of the new vertex contains any center vertex c whose degree is higher, the new vertex becomes a satellite vertex of c. The difficult case that destroys the correctness of the star cover is when the new vertex is adjacent to a collection of star centers, each of whose degree is lower than that of the new vertex. In this situation, the star structure already in place has to be modified to assign the new vertex as a star center. The satellite vertices in the stars that are broken as a result have to be re-evaluated.

4.1 The on-line star algorithm

Motivated by the intuition in the previous section, we now describe an on-line algorithm for incrementally computing star covers of dynamic graphs. The algorithm is shown in Figure 4. This algorithm uses a data structure to efficiently maintain the star covers of an undirected graph $G = (V, E)$. For each vertex $v \in V$, we maintain the following data.

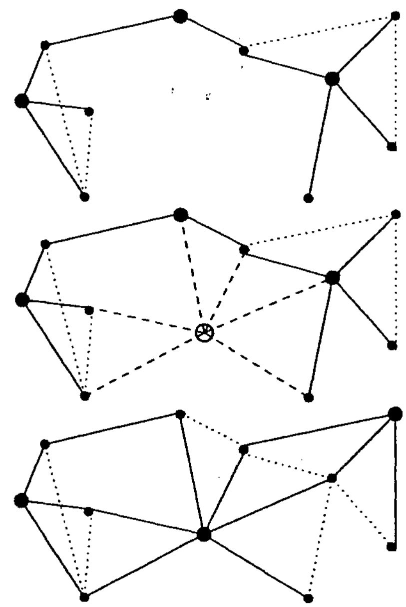


Figure 3: This figure shows the star cover change after the insertion of a new vertex. The larger-radius disks denote star centers, the other disks denote satellite vertices. The star edges are denoted by solid lines. The inter-satellite edges are denoted by dotted lines. The top figure shows an initial graph and its star cover. The middle figure shows the graph after the insertion of a new document. The bottom figure shows the star cover of the new graph.

$v.type$	satellite or center
$v.degree$	degree of v
$v.adj$	list of adjacent vertices
$v.centers$	list of adjacent centers
$v.inQ$	flag specifying if v being processed

Note that while $v.type$ can be inferred from $v.centers$ and $v.degree$ can be inferred from $v.adj$, it will be convenient to have all five pieces of data in the algorithm. Let α be a vertex to be added to G, and let L be the list of vertices in G which are adjacent to α. The algorithm in Figure 4 will appropriately update the star cover of G.

The algorithm maintains a priority queue Q of vertices not yet correctly placed in the star cover. When a star is broken, its center and satellites are placed in Q.

The on-line star cover algorithm is more complex than its off-line counterpart. We devote the rest of this section to proving that the algorithm is correct and to analyzing its expected running time.

```
UPDATE(α, L)
 1  α.type ← satellite
 2  α.degree ← 0
 3  α.adj ← ∅
 4  α.centers ← ∅
 5  forall β in L
 6     α.degree ← α.degree + 1
 7     β.degree ← β.degree + 1
 8     INSERT(β, α.adj)
 9     INSERT(α, β.adj)
10     if (β.type = center)
11        INSERT(β, α.centers)
12     else
13        β.inQ ← true
14        ENQUEUE(β, Q)
15     endif
16  endfor
17  α.inQ ← true
18  ENQUEUE(α, Q)
19  while (Q ≠ ∅)
20     φ ← EXTRACTMAX(Q)
21     if (φ.centers = ∅)
22        φ.type ← center
23        forall β in φ.adj
24           INSERT(φ, β.centers)
25        endfor
26     else
27        if (∀δ ∈ φ.centers, δ.degree < φ.degree)
28           φ.type ← center
29           forall β in φ.adj
30              INSERT(φ, β.centers)
31           endfor
32           forall δ in φ.centers
33              δ.type ← satellite
34              forall μ in δ.adj
35                 DELETE(δ, μ.centers)
36                 if (μ.inQ = false)
37                    μ.inQ ← true
38                    ENQUEUE(μ, Q)
39                 endif
40              endfor
41           endfor
42        endif
43     endif
44     φ.inQ ← false
45  endwhile
```

Figure 4: The on-line star algorithm for clustering.

4.2 Correctness

In this section we show that the on-line algorithm is correct by proving that it produces the same star cover as the off-line algorithm, when the off-line algorithm is run on the final graph considered by the on-line algorithm. Before we state the result, we note that the off-line star algorithm does not produce a unique cover. When there are several vertices of the same highest degree, the algorithm arbitrarily chooses one of them as the next star center. We will show that the cover produced by the on-line star algorithm is the same as one of the covers that can be produced by the off-line algorithm

THEOREM 4.1. *The cover generated by the on-line star algorithm when $G_\sigma = (V, E_\sigma)$ is constructed incrementally (by inserting its vertices one at a time) is identical to some legal cover generated by the off-line star algorithm on G_σ.*

Proof. We can view a star cover of G_σ as a correct assignment of types (that is, "center" or "satellite") to the vertices of G_σ. The off-line star algorithm assigns correct types to the vertices of G_σ. We will prove the correctness of the on-line star cover by induction. The induction invariant is that at all times, the types of vertices not in Q are correct, *assuming* that the true type of vertices in Q is "satellite." This would imply that when Q is empty, all vertices are assigned a correct type, and thus the star cover is correct.

The invariant is true initially: as the type of the new node α is unknown and α is in Q; the type of all the satellite neighbors of α are unknown and these neighbors are in Q; and all the other vertices have correct types from the original cover, assuming that the nodes in the queue are correctly satellite. We now show that the induction invariant is maintained throughout the algorithm. Consider Figure 4. The first thing to note is that the type of all the vertices in Q is "satellite"; statements 14, 18 and 33 enqueue satellite vertices. We now argue that every time a vertex ϕ of highest degree is pulled out of Q, it is assigned a correct type. When ϕ has no centers on its adjacency list, its type should be "center" (which is assigned correctly by statement 22). When ϕ is adjacent to star centers δ_i, each of which has a lower degree that ϕ, the correct type for ϕ is "center" (statement 28). This action has a side effect: all δ_i cease to be star centers and thus get enqueued for further evaluation (statements 32-39). Otherwise, the correct type for ϕ is the default "satellite". Since ϕ was extracted from Q and all vertices in Q are satellites, the type of ϕ is correct in this case as well.

To complete the argument, what remains to be shown is that eventually the queue Q becomes empty. The termination of the while loop at statement 19 in Figure 4 is guaranteed by the following result.

LEMMA 4.1. *The degree of the stars broken by the on-line star algorithm is strictly decreasing.*

The lemma is equivalent to the following statement: node ϕ in Q has the potential of becoming a star center and has the capability of adding new nodes γ to Q that can become stars of degree strictly less than the degree of node ϕ.

Suppose ϕ becomes a new star center. We show than its satellite neighbors γ cannot become star centers. Two cases arise. (Case 1) γ_i is not a star center because its degree is smaller than the degree of the new star center that covers ϕ in the new cover. (Case 2) γ_i is not a star center because it is a satellite of a much larger star, so its degree is larger than the degree of the new star that covers ϕ. But this condition still holds after making the new star. This completes the proof sketch for the termination lemma and it follows that the types assigned by the on-line algorithm are correct; in other words, that there exists an off-line algorithm that produces the same cover.

4.3 Running Time Analysis and Experimental Results

In this section, we argue that the running time of the on-line star algorithm is efficient in practice, asymptotically matching the running time of the off-line star algorithm ($\Theta(V+E)$) to within lower order factors. We first note, however, that there exist worst-case thresholded similarity graphs G_σ and corresponding vertex insertion sequences which cause the on-line star algorithm to run in $\Theta(V^2)$ time.[6] These graphs and insertion sequences rarely arise in practice though. An analysis more closely modeling practice is the random graph model in which G_σ is a random graph and the insertion sequence is random. In this model, the *expected* running time of the on-line star algorithm can be determined.

In the sections that follow, we first give intuition for the expected running time of the on-line star algorithm. In subsequent sections, we give experimental results showing that the on-line star algorithm is quite efficient with respect to both random data and a large collection of real documents.

4.3.1 Intuition

We have implemented the on-line star algorithm using a heap for the priority queue and simple linked lists for the various lists required. The time required to insert a new vertex and associated edges into a thresholded similarity graph and to appropriately update the star cover is largely governed by the number of stars that are broken during the update, since breaking stars requires inserting new elements into the priority queue. In practice, very few stars are broken during any given update. This is due partly to the fact that relatively few stars exist at any given time (as compared to the number of vertices or edges in the thresholded similarity graph) and partly to the fact that the likelihood of breaking any individual star is also small. We begin with the former, noting that the number of stars expected to cover a random graph $G_{n,p}$ is only $\Theta(\log n)$.

THEOREM 4.2. *The expected size of the star cover for* $G_{n,p}$ *is* $\frac{\log n}{\log(\frac{1}{1-p})}$.

Proof. The star cover algorithm is greedy: it iterates by selecting the highest degree vertex not yet covered as a star center and marking this node and all its adjacent vertices as covered. Each iteration creates a new star. We will argue that the number of iterations is $\frac{\log n}{\log(\frac{1}{1-p})}$. The argument relies on the random graph model described in Section 2.1.

Let $G_{n,p}$ correspond to a random graph on n vertices where each edge exists with probability p. The degree of each vertex of G is distributed binomially: $Pr[\deg = k] = bin(k; n-1, p) = \binom{n-1}{k}\ p^k(1-p)^{n-1-k}$. The mean of this distribution is $\mu = (n-1)p$ and its variance is $\sigma = \sqrt{(n-1)p(1-p)}$. Note that while the the degrees of the vertices do exhibit some dependence, for practical purposes they can be considered independent [Bol95]. This means that on average, each star covers $(n-1)p+1 \geq np$ vertices.[7] Since the np vertices covered by each star are randomly chosen, there will be some overlap between the star covers. Each new star leaves uncovered a $(1-p)$ fraction of the previously uncovered vertices. In other words, after the first iteration, $(1-p)n$ vertices remain uncovered. After i iterations, $(1-p)^i n$ vertices remain uncovered. The algorithm terminates when all the vertices are covered, or $(1-p)^i n < 1$. By taking logs of both sides of this inequality, it follows that $i > \frac{\log n}{\log(\frac{1}{1-p})}$ is sufficient.

Thus, the number of stars is expected to be relatively small. Furthermore, the probability any individual star will be broken is quite small as well. A star can only be broken if the star center has the same degree as one of its associated satellite vertices and if the vertex being added to the graph is connected to that satellite but not to the star center.[8] In practice, the expected number of stars broken during an update is a small constant even for graphs containing thousands of vertices (though asymptotically it is certainly a slowly

[6] An example is a graph consisting of two connected vertices A and B of very high but identical degree (not both of which can be star centers) and an insertion sequence which causes the "local" center to repeatedly switch between A and B.

[7] The star covers its center and $(n-1)p$ satellites.

[8] Once a star is broken during an update, however, other stars can be broken in different ways via a cascading effect.

growing function of n). In Figure 5, we give experimental results showing that the total number of stars broken during runs on two different types of data is roughly a linear function of the number of vertices; thus, the expected number of stars broken during any given update is roughly a constant (or more likely a slowly growing function of n).

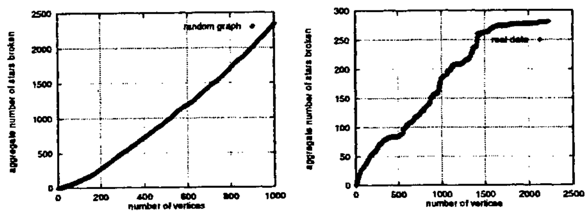


Figure 5: (The dependence of the number of broken stars on the number of vertices in a random graph (left) and for text data (right).

The time to break a star is roughly proportional to its size (the degree of its associated star center), and since the degrees of all vertices are expected to be similar in distribution ($bin(k; n-1, p)$), this is on the order of the number of edges being inserted into the graph. Since only a constant number of stars are expected to be broken, the expected time to perform an update will be roughly proportional to the number of edges inserted in the graph during the update. Thus, the total time to perform n updates should be roughly proportional to the total number of edges in the final graph. In the sections that follow, we give experimental results which confirm this fact.

4.3.2 Experimental results

We have experimented with the on-line clustering algorithm in two scenarios. The first type of data matches our random graph model and consists of random similarity graphs. While this type of data is useful as a benchmark for the running time of the algorithm, it does not satisfy the geometric constraints of the vector space model. We also conducted experiments using real data from the TREC collection[9] as a second type of benchmark for the algorithm.

We now detail our data generation procedure and the experimental running time of the on-line star algorithm on each data type.

[9]TREC is the annual text retrieval conference. TREC is organized as a competition. Each participant is given on the order of 5 gigabytes of data and a standard set of queries on which to test their systems. The results and the system descriptions are presented as papers at the TREC conference.

Generating Random Data. We ran the on-line star cover algorithm on a random graph with 1000 nodes. The edges in this graph were inserted randomly with probability $p = 0.2$. The on-line algorithm was run 30 times. Each time, the vertices of the random graph were inserted in random order. The results were averaged over the 30 experiments. Figure 6 shows the data from these experiments. Note that the the running time is roughly linear in the number of edges in the graph, and we can see the effects of lower order terms.

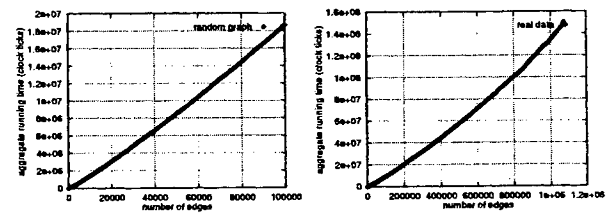


Figure 6: This figure shows the dependence of the running time of the on-line star algorithm on the number of edges in a random graph (left) and for text data (right).

Experiments with real data. We ran the on-line star cover algorithm on a document collection that consists of a slice of TREC documents augmented with our department's technical reports. The resulting collection consists of 2224 documents. We ran four experiments. Each time we set a different threshold and added the similarity graph nodes in random order. The results of these experiments were averaged and the running time measurements appear to be linear in the number of edges of the similarity graph. Figure 6 shows the data from these experiments. Note that the the running time is roughly linear in the number of edges in the graph, and we can see the effects of lower order terms.

Comparion between Star, Single Link, and Average Link on TREC Data. We have implemented a system for organizing information that uses the star algorithm. Figure 7 shows the user interface to this system [APR97]. In order to evaluate the performance of our system, we tested the star algorithm against two widely used clustering algorithms in IR: the single link method [Rij79] and the average link method [Voo85]. We used data from the TREC-6 conference as our testing medium. The TREC collection contains a very large set of documents of which 21,694 have been ascribed relevance data with respect to 47 topics. These 21,694 documents were partitioned into 22 separate subcollections of approximately 1,000 documents each for 22 rounds of the following experiment. For each of the 47 topics the given collection of

documents was clustered with each of the three algorithms and the best cluster was returned. To measure the quality of a cluster, we use the common E measure [Rij79] defined as: $E(p,r) = 1 - 2/(1/p + 1/r)$, where p and r are the standard *precision* and *recall* of the cluster with respect to the set of documents relevant to the topic.[10] It is worthwhile to note that in viewing data comparing two clustering methods, *lower* $E(p,r)$ values correspond to *better* performance. Averaging over all 22 experiments, we find that the mean $(p, r, E(p,r))$ values for star, average-link and single-link are $(0.77, 0.54, 0.36)$, $(0.83, 0.44, 0.42)$ and $(0.84, 0.41, 0.45)$, respectively. Thus, the star algorithm represents a 16.6% improvement in performance with respect to average-link and an 25% improvement with respect to single-link.

Figure 8 shows the detailed E measures for the star algorithm vs. the single link algorithms and for the star algorithm vs. the average link algorithm. We collected all the topic clusters computed in this experiment. We sorted the clusters produced by the star algorithm according to their E-values. We plotted the E value for the coresponding cluster computed by the single link algorithm (see the oscillating line in Figure 8–left) and for the average link algorithm (see the oscillating line in Figure 8–right). We note that the star algorithm outperforms both the single link algorithm and the average link algorithm, because the E values for the star clusters are almost everywhere lower than the corresponding values for the other two algorithms. Note that not all topics are present in all 22 experiments, which is why we have only approximately 500 clusters in these graphs.

Our experiments show that in general, the star algorithm outperforms single link by 25% and that it outperforms average link by 16.6%. We repeated this experiment on the same data, using one collection only (of 21,694 documents), and obtained similar results.[11] These improvements are significant for text applications. Considering that the star algorithm outperforms the average link algorithm, it is easier to implement than the average link algorithm, it can be used as an on-line algorithm, and it runs much faster, these experiments provide support for using the star algorithm in clustering and off-line information organization.

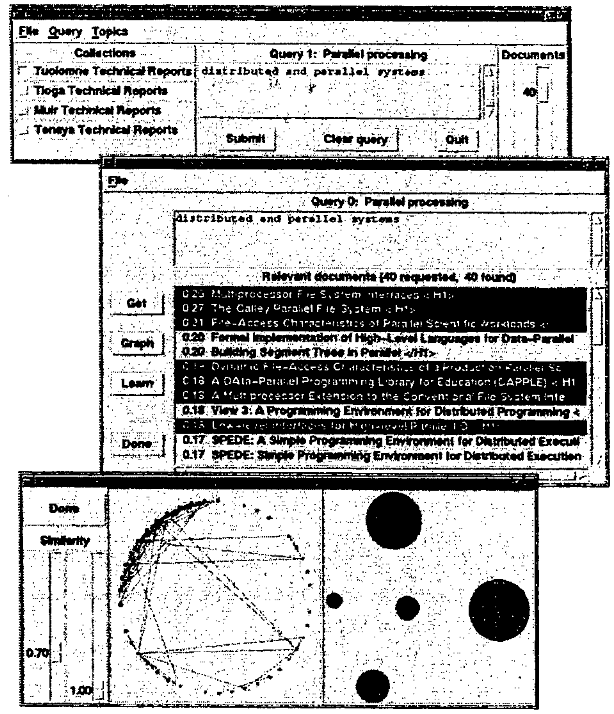


Figure 7: This is a screen snapshot from a clustering experiment. The top window is the query window. The middle window consists of a ranked list of documents that were retrieved in response to the user query. The user my select "get" to fetch a document or "graph" to request a graphical visualization of the clusters as in the bottom window. The left graph displays all the documents as dots around a circle. Clusters are separated by gaps. The edges denote pairs of documents whose similarity falls between the slider parameters. The right graph displays all the clusters as disks. The radius of a disk is proportional to the size of the cluster. The distance between the disks is proportional to the similarity distance between the clusters.

5 Discussion

We presented and analyzed an off-line clustering algorithm for static information organization and an on-line clustering algorithm for dynamic information organization. We discussed the random graph model for analyzing these algorithms and showed that in this model, the algorithms have expected running time that is roughly linear in the number of edges. The data we gathered from experimenting with these algorithms provides support for the validity of our model and analysis. The empirical tests show that both algorithms have an asymptotic linear time performance in the number of edges in the graph. In addition, both algorithms are simple and easy to implement. We believe that the fast running

[10] Precision is the fraction of returned documents that are correct. Recall is the fraction correct documents that are returned.

[11] The precision, recall, and E values for star, average link, and single link were (.53, .32, .61), (.63, .25, .64), and (.66, .20, .70), respectively. We note that the E measures are worse for all three algorithms on this larger collection and that the star algorithm outperforms average link by 10.3% and single link by 20.7%.

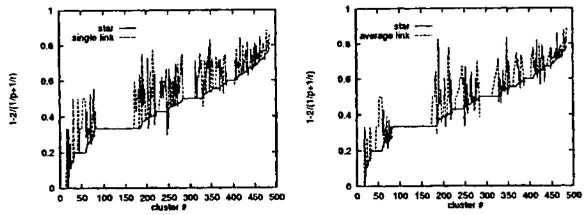


Figure 8: This figure shows the E measure for the star clustering algorithm vs. the single link clustering algorithm (left) and the star algorithm vs. the average link algorithm (right). The y axis shows the E measure. The x axis shows the cluster number. Clusters have been sorted according to the E value for the star algorithm.

time and the ease of implementation make these algorithms very practical candidates for use in automatically organizing digital libraries.

This work departs from previous clustering algorithms used in information retrieval that use a fixed number of clusters for partitioning the space. Since the number of clusters produced by our algorithms is given by the underlying topic structure in the information system, our clusters are dense and accurate. Our work extends previous results [HP96] that support using clustering for browsing applications and presents positive evidence for the cluster hypothesis. In [APR97], we argue that by using a clustering algorithm that guarantees the cluster quality through separation of dissimilar documents and aggregation of similar documents, clustering is beneficial for information retrieval tasks that require high precision and high recall. Precision-recall are the standard measurements for the performance of an information retrieval algorithm [Sal89].

References

[AB84] M. Aldenderfer and R. Blashfield, *Cluster Analysis*, Sage, Beverly Hills, 1984.

[APR97] J. Aslam, K. Pelekhov, and D. Rus, Generating, visualizing, and evaluating high-accuracy clusters for information organization, in *Principles of Digital Document Processing*, eds. C. Nicholas, LNCS Springer Verlag 1998.

[Bol95] B. Bollobás, *Random Graphs*, Academic Press, London, 1995.

[Can93] F. Can, Incremental clustering for dynamic information processing, in *ACM Transactions on Information Systems*, no. 11, pp143-164, 1993.

[CCFM97] M. Charikar, C. Chekuri, T. Feder, and R. Motwani, Incremental clustering and dynamic information retrieval, in *Proceedings of the* 29^{th} *Symposium on Theory of Computing*, 1997.

[Cro80] W. B. Croft. A model of cluster searching based on classification. *Information Systems*, 5:189-195, 1980.

[Cro77] W. B. Croft. Clustering large files of documents using the single-link method. *Journal of the American Society for Information Science*, pp189-195, November 1977.

[CKP93] D. Cutting, D. Karger, and J. Pedersen. Constant interaction-time scatter/gather browsing of very large document collections. In *Proceedings of the* 16^{th} *SIGIR*, 1993.

[FG88] T. Feder and D. Greene, Optimal algorithms for approximate clustering, in *Proceedings of the* 20^{th} *Symposium on Theory of Computing*, pp 434-444, 1988.

[HP96] M. Hearst and J. Pedersen. Reexamining the cluster hypothesis: Scatter/Gather on Retrieval Results. In *Proceedings of the* 19^{th} *SIGIR*, 1996.

[HS86] D. Hochbaum and D. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *Journal of the ACM*, no. 33, pp533-550, 1986.

[JD88] A. Jain and R. Dubes. *Algorithms for Clustering Data*, Prentice Hall 1988.

[JR71] N. Jardine and C.J. van Rijsbergen. The use of hierarchical clustering in information retrieval, 7:217-240, 1971.

[KP93] G. Kortsarz and D. Peleg. On choosing a dense subgraph. In *Proceedings of the 34th Annual Symposium on Foundations of Computer Science (FOCS)*, 1993.

[LY94] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *Journal of the ACM* 41, 960–981, 1994.

[Rij79] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[Sal89] G. Salton. *Automatic Text Processing: the transformation, analysis, and retrieval of information by computer*, Addison-Wesley, 1989.

[Sal91] G. Salton. The Smart document retrieval project. In *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 356-358.

[Tur90] H. Turtle. Inference networks for document retrieval. PhD thesis. University of Massachusetts, Amherst, 1990.

[Voo85] E. Voorhees. The cluster hypothesis revisited. In *Proceedings of the* 8^{th} *SIGIR*, pp 95-104, 1985.

[Wil88] P. Willett. Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management*, 24:(5):577-597, 1988.

[Wor71] S. Worona. Query clustering in a large document space. In Ed. G. Salton, *The SMART Retrieval System*, pp 298-310. Prentice-Hall, 1971.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *Proceedings of the Eight Annual Structure in Complexity Theory Conference*, IEEE Computer Society, 305–312, 1993.

Scalable Information Organization*

Javed Aslam, Fred Reiss, and Daniela Rus
Department of Computer Science
Dartmouth College
Hanover, NH 03755 USA
{jaa, frr, rus}@cs.dartmouth.edu

Abstract

We present three scalable extensions of the star algorithm for information organization that use sampling. The star algorithm organizes a document collection into clusters that are naturally induced by the topic structure of collection, via a computationally efficient cover by dense subgraphs. We also provide supporting data from extensive experiments.

1 Introduction

Our goal is to develop a completely automated information organization system for digital libraries, automated tools for librarians to classify this information, automatic tools to create reference pointers into such collections, and automated tools that allow users to locate information effectively.

We focus on static and dynamic digital collections of unstructured text. We consider the problem of determining the topic structure of text data, without *a priori* knowledge of the number of topics in the data or any other information about their composition. We assume that the collections may be static (for example, digital legacy collections) or dynamic (for example, news wires). We look to discover hierarchies of topics and subtopics in such text collections. Thus, we develop clustering algorithms that can be used in off-line, on-line, and hierarchical mode. We wish for these algorithms to be fast, scalable, accurate, and to discover the naturally occurring topics in the collection. In our previous work (Aslam et al, 1998; Aslam et al, 1999), we proposed an off-line and an on-line approach based on graph theory. Our algorithms, called the *star clustering* algorithms, compute clusters induced by the *natural* topic structure of the space. Thus, this work is different than previous work in using clustering to organize information (Cutting et al, 1993; Charikar et al, 1997) in that we do not impose the constraint to use a fixed number of clusters. This previous work argues that the star algorithm is simple, efficient, can be used in off-line as well as on-line mode, and it outperforms existing clustering algorithms such as single link, average link, and k-means. In this paper we consider scalability issues in developing an information organization system. We present three different scalable extensions to the star algorithm and show data from extensive experiments.

2 Related Work

There has been extensive research on clustering and applications to many domains (Everitt, 1993; Mirkin 1996; Silverstein and Pedersen 1997; Sibson, 1973; Worona, 1971). For a good overview see (Jain and

Dubes, 1988). For a good overview of using clustering in Information Retrieval (IR) see (Willett, 1988). The use of clustering in IR was mostly driven by *the cluster hypothesis* (Rijsbergen, 1979) which states that relevant documents tend to be more closely related to each other than to non-relevant documents. Efforts have been made to find whether the cluster hypothesis is valid. Voorhees (Voorhees, 1985) discusses a way of evaluating whether the cluster hypothesis holds and shows negative results. Croft (Croft, 1080) describes a method for bottom-up cluster search that could be shown to outperform a full ranking system for the Cranfield collection. In (Jardine and van Rijsbergen, 1971) Jardine and van Rijsbergen show some evidence that search results could be improved by clustering. Hearst and Pedersen (Hearst and Pedersen, 1996) re-examine the cluster hypothesis by focusing on the Scatter/Gather system (Cutting et al, 1993) and conclude that it holds for browsing tasks.

Systems like Scatter/Gather (Cutting et al, 1993) provide a mechanism for user-driven organization of data in a fixed number of clusters, but the users need to be in the loop and the computed clusters do not have accuracy guarantees. Scatter/Gather uses fractionation to compute nearest-neighbor clusters. Charika et al. (Charikar et al, 1997) consider a dynamic clustering algorithm to partition a collection of text documents into a *fixed number* of clusters. Since in dynamic information systems the number of topics is not known *a priori*, a fixed number of clusters cannot generate a natural partition of the information.

3 Background: The Star Algorithm for Information Organization

For any threshold σ:

1. Let $G_\sigma = (V, E_\sigma)$ where $E_\sigma = \{e : w(e) \geq \sigma\}$.
2. Let each vertex in G_σ initially be *unmarked*.
3. Calculate the degree of each vertex $v \in V$.
4. Let the highest degree unmarked vertex be a star center, and construct a cluster from the star center and its associated satellite vertices. Mark each node in the newly constructed cluster.
5. Repeat step 4 until all nodes are marked.
6. Represent each cluster by the document corresponding to its associated star center.

Figure 1: The star algorithm

To compute accurate topic clusters, one possibility is to formalize clustering as covering similarity graphs by cliques. A clique cover will guarantee that its documents are strongly related to each other. Covering by cliques is NP-complete, and thus intractable for large document collections. Unfortunately, it has also been shown that the problem cannot even be approximated in polynomial time (Zuckerman, 1993). We instead propose using a cover by *dense subgraphs* that are *star-shaped* and that can be computed *off-line* for static data and *on-line* for dynamic data. What we lose in intra-cluster similarity guarantees, we gain in computational efficiency.

We represent the document collection as a complete similarity graph, where the vertices correspond to documents and the edges are weighted by a similarity measure. We have used two measures: the cosine metric and an information-theoretic metric.

To compute accurate topic clusters, we create a thresholded similarity graph, where the thresholding

parameter is given by the smallest similarity we would like to have between any documents within a topic. We then approximate a clique cover of this graph by covering the associated thresholded similarity graph with *star-shaped subgraphs*. A star-shaped subgraph on $m + 1$ vertices consists of a single *star center* and m *satellite vertices*, where there exist edges between the star center and each of the satellite vertices. A greedy algorithm (see Figure 1) computes this cover for static collections. In (Aslam et al, 1998; Aslam et al, 1999) we show an on-line version of this algorithm that supports information organization in dynamic collection.

Star-graph covers are interesting because they provide accuracy guarantees on the computed topics. By investigating the geometry of the problem, we can derive a *lower bound* on the similarity between satellite vertices as well as provide a formula ($\cos\gamma \geq \cos\alpha_1 \cos\alpha_2 + \frac{\sigma}{1+\sigma} \sin\alpha_1 \sin\alpha_2$, where α_1 and α_2 correspond to the similarity between the center and the two satellites and σ is the similarity threshold) for the *expected* similarity between satellite vertices using the cosine metric. This formula predicts that the pairwise similarity between satellite vertices in a star-shaped subgraph is high, and together with empirical evidence supporting this formula (Aslam et al, 1998).

4 Scalable Extensions for the Star Algorithm

For any threshold σ:

1. Let D be a set of n documents sorted in random order in an array.
2. Let s be the sample size.
3. Compute a Star Cover for $D[1..s]$ and let C be the list of star centers of this cover.
4. For each document $D[i]$ in $D[s + 1..n]$
   - For each cluster $C[j]$ in C: if $similarity(D[i], C[j]) > \sigma$ insert $D[i]$ in $C[j]$
   - If $D[i]$ was not inserted in any existing cluster, create a new cluster with $D[i]$ as a center and add this cluster to C.

Figure 2: The sampled star algorithm.

In this section we present three extensions to the star algorithm that optimize its performance. The three algorithms compute approximations to the star cluster but optimize on the size of the similarity matrix used and on the time required to generate it.

Both of the off-line and on-line versions of the star algorithm rely on the existence of the similarity matrix. Similarity matrices can get very large: for a document set with n documents the similarity matrix is $O(n^2)$ space data structure. However, this operation, which takes $O(n^2)$ time to compute[1], is much more expensive than the basic cost of the star clustering algorithm, which is approximately $O(n)$ time. Thus, it is clear that the similarity matrix is a bottleneck. Computing this matrix is a one-time pre-processing operation. However, the data structure has to be available on a permanent basis. For these reasons, we now investigate several methods to improve on the similarity matrix bottleneck.

[1]Note that the actual time is $O(n^2)$ times the cost of a vector dot product; because the vectors are sparse, this translates into $O(n^2)$ with a high constant.

4.1 Sampled Stars

The first approximation algorithm uses sampling to compute the similarity matrix and is called the *sampled star* algorithm (see Figure 2). The basic idea behind this algorithm is to create a sample of the document collection that is much smaller than the actual collection. This sample can then be used to compute a complete Star Clustering, using the off-line star algorithm. For this small set, the computation of the similarity matrix is much faster. Finally, the rest of the documents can be inserted in the resulting clusters fast by comparing each document against the existing star centers only. Documents that are not close enough to any existing star centers (that is, all distances to existing star centers are below the threshold) form new clusters. Alternatively, the additional documents can be inserted in the cluster structure using the on-line star algorithm.

4.2 Linear-space Stars

For any threshold σ:

1. Let D be a set of n documents, p a desired probability, and σ a threshold.
2. Let $C = \emptyset$ denote the desired clustering.
3. Select a sample S of pairs of documents (d_1, d_2) from D
4. For each pair (d_1, d_2) in S if the dot product between $(d_1, d_2) > \sigma$ increase the degrees of d_1 and d_2.
5. Sort D in descending order by degree.
6. Find and mark all the star centers by examining one-by-one the sorted D.
7. For $i = 1$ to n insert d_i into all possible star centers.

Figure 3: The linear space sampled star algorithm.

The sampled star algorithm provides a more effective way to compute the overall clustering of a document set but even this algorithm requires the computation of a complete similarity matrix (which is smaller than the original matrix). An additional optimization is to remove entirely the similarity matrix. The key information used by the star algorithm is the degree of the nodes in the thresholded similarity graph. This information can be represented in an array. A trivial algorithm for generating the array is to compare every document against every other document and count the number of vector products about the threshold. Note that this method reduces significantly the space requirements but still necessitates $O(n^2)$ time to generate, where n is the number of documents. An alternative is to compute the vertex degrees approximately, using sampling. For each document, we first generate a sample of documents to be used for comparison. A dot product is computed between the document and each member of the sample set. The degree of the document vertex is given by the number of dot products that are above the threshold. Figure 3 summarizes this algorithm.

4.3 Distributed Stars

Another bottleneck for the star algorithm comes up in Internet applications, such as organizing data collected from various sites and databases by topic. Consider a task in which several databases are

For any threshold σ:

1. Let D be a set of n documents. Divide D into k disjoint sets $D_1 \ldots D_k$.
2. Run the Star algorithm on k separate machines to produce the star clusterings $C_1 \ldots C_k$.
3. Let $c_1 \ldots c_j$ be the set of star centers in all the star covers.
4. Run the Star algorithm on the set of documents $c_1 \ldots c_j$.
5. If two star centers are placed in the same cluster in the previous step, merge their clusters using a union operation.

Figure 4: The distributed star algorithm.

queried with the same question. The documents returned by these queries are to be fused and presented to the user in a coherent picture. One approach is to run the queries, download all documents, and organize the entire collection at the user site using the star algorithm. An alternative approach is to run the queries, organize the search results at the location of the database, and then merge these results on the user machine. This second alternative has several advantages: (1) the star algorithm can be run in parallel, which provides a speedup; (2) the document transfer operation can also be parallelized[2]; and (3) the local topic organizations can be viewed as a way of compressing the documents, can be used to generate the merged topics in the distributed collection, and can be transfered much faster than the actual documents to the user's machine.

For these reasons, we describe a third approximation of the star algorithm called *the distributed star* algorithm, which is useful especially when the document collection is very large. The distributed star algorithm provides parallelism and is based on a "divide and conquer" approach. The document collection is partitioned into several disjoint sets. The sets are clustered separately and the resulting clusters are then merged. Figure 4 shows the details of this algorithm. Note that for this version of the algorithm, the off-line Star algorithm can be replaced with the Sampled Star algorithm or with the Linear Space Star algorithm.

4.4 Experiments and Evaluations

We devised two experiments for the purpose of testing our algorithms on real-world data. Because we were limited by computer memory, we focused the experiments on the Linear Space Sampled Star algorithm (see Figure 3) which was introduced to optimize both time performance and space requirements.

In our first experiment, we ran the Linear Space Sampled Star algorithm on a 50000- document subset of the TREC volume 1 corpus at various sample sizes. We compared the output of the Linear Space Sampled Star algorithm with sampling to its output without sampling, and show these results in Figure 5. Note that when sampling is not used, the the Linear Space Sampled Star algorithm produces the same output as the Star algorithm.

To measure the difference between the outputs of the two algorithms, we calculated an aggregate precision and recall for each sample size as follows. For each cluster x in the output of the sampled algorithm, we calculated the precision and recall of the documents in x against each cluster in the output of the unsampled algorithm. We then determined the cluster y in the output of the unsampled algorithm that

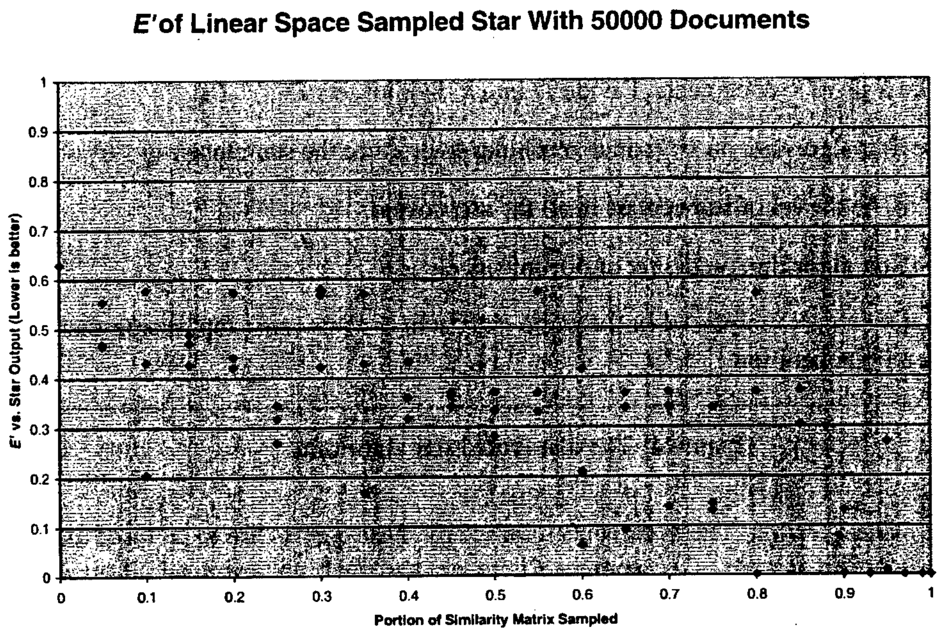


Figure 5: The effects of the sample size on the quality of clusters obtained using the Linear Space Sampled Star algorithm. The x-axis shows the sample size. The y-axis shows the aggregate E-measure computed relative to the star algorithm. The smaller the E-value is, the better the performance is. The experiment was done with a TREC subset of 50000 documents.

minimizes van Rijsbergen's (Rijsbergen, 1979) evaluation measure

$$E(p,r) = 1 - \frac{2}{1/p + 1/r}$$

where p and r are the standard *precision* and *recall* of the cluster with respect to the set of documents relevant to the topic. Finally, we calculated a weighted average E' of the E-values calculated previously, weighting each E value by the number of documents in the associated cluster. Figure 1 shows the results of this analysis. With larger samples, the sampled algorithm generally produced the exact same results as the algorithm that did not use sampling. As the portion of the similarity matrix sampled decreased, the results of the sampled algorithm deviated increasingly from those of the unsampled algorithm.

Our subsequent analyses sought to determine whether the divergent output of the sampled algorithm was inferior to the output of the unsampled algorithm. The original purpose of the Star algorithm was to calculate a cover of the input documents using as few star-shaped clusters as possible (Aslam et al, 1998; Aslam et al, 1999). The Linear Space Sampled Star algorithm also generates a cover of the input documents with star-shaped clusters, so we compared the number of clusters in the algorithm's output at varying sample sizes to the number of clusters in the output of the unsampled algorithm (see Figure 6). Surprisingly, even with samples as small as 5%, the number of clusters output by the sampled algorithm was never more than five percent larger than the number of clusters that the unsampled algorithm generated. In fact, the sampled algorithm generally covered the corpus with fewer star-shaped clusters than unsampled algorithm did.

Our second experiment compared the output of the Linear Space Sampled Star algorithm against categorization decisions made by humans. Specifically, the algorithm was run on 4925 documents from the FBIS corpus that had been labeled by humans with one or more of 47 different categories. We repeated the precision/recall analysis of the first experiment, using the 47 categories in the place of the output of the unsampled Star algorithm. As with the previous experiment, samples as small as 1% produced results

Number of Clusters for Linear Space Sampled Star with 50000 Documents

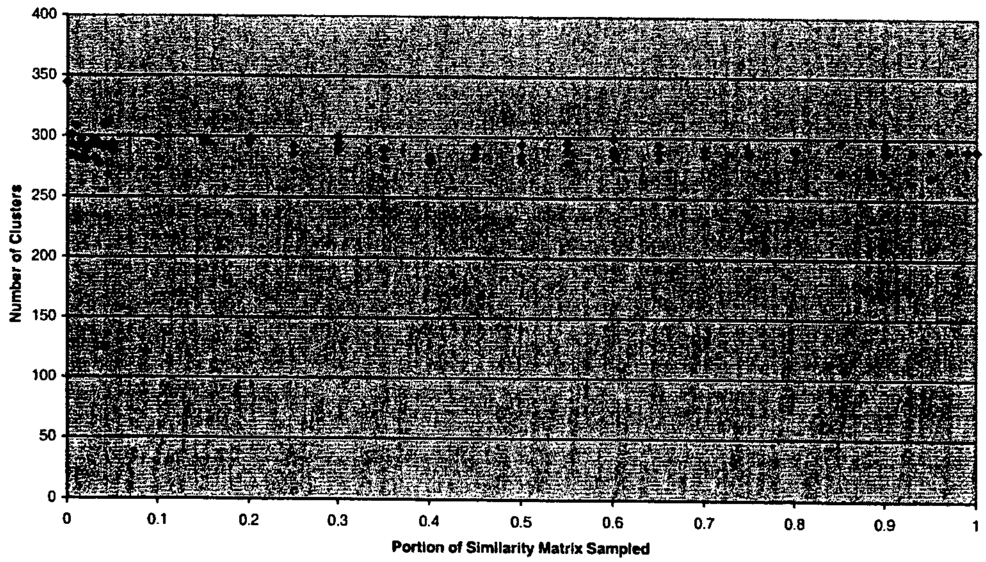


Figure 6: The effect of sampling on the number of clusters generates. The x-axis shows the sampling size. The y-axis shows the ratio between the number of clusters generated by the Linear Space Sampled Star algorithm to the number of clusters generated by the Star algorithm. We observe that sampling does not affect much the number of clusters discovered in the collection. The experiment was done with a TREC subset of 50000 documents.

comparable to a 100% sample (See Figure 7).

Overall, our experiments indicated that the Linear Space Sampled Star algorithm generates output comparable in quality to that of the Star algorithm, but uses considerably fewer CPU and memory resources. Both of our implementations of the Linear Space Star algorithm required only 81 megabytes of memory to process 50000 documents, 73 megabytes of which was only used to store the vector representations of the documents. On the other hand, an implementation of the Star algorithm that uses a sparse thresholded similarity matrix would require approximately 2.5 gigabytes of memory for 50000 documents, and a complete similarity matrix stored in a double-precision floating-point array would require 18.6 gigabytes of memory. The gains in performance due to sampling were similarly significant. Figure 8 shows the amount of time that the Linear Space Sampled Star algorithm requires to process 50000 documents at varying sample sizes. These times were measured on a 250 MHz. MIPS R10000 and do not include the time required to parse the documents. We found the running time of the algorithm to be almost directly proportional to the size of the sample. At sample sizes of less than 5%, the Linear Space Sampled Star algorithm organized documents at an average rate comparable to the bandwidth of most Internet connections (See Figure 9). Tests comparing the Star algorithm with the Linear Space Sampled Star algorithm on smaller data sets indicated that the overhead of sampling and reducing memory requirements result in an increase in running time of less than 5%.

Finally, we have conducted a small experiment on 1000 TREC documents to study the performance of the Distributed Star algorithm Figure 10 shows the accuracy of the distributed star algorithm relative to the off-line star algorithm. We note that when the number of computers is the same as the number of documents, Step 4 of the Distributed Star Algorithm (Figure 4) performs a star clustering of the entire collection. The same is true when there is a single machine. The greatest degree of parallelism and distribution is achieved when the number of machines is $\sqrt{m}$, where m is the number of machines in the system. For this experiment, $m = 1000$ and $\sqrt{m}$ is approximately 32. The experiment shows that the

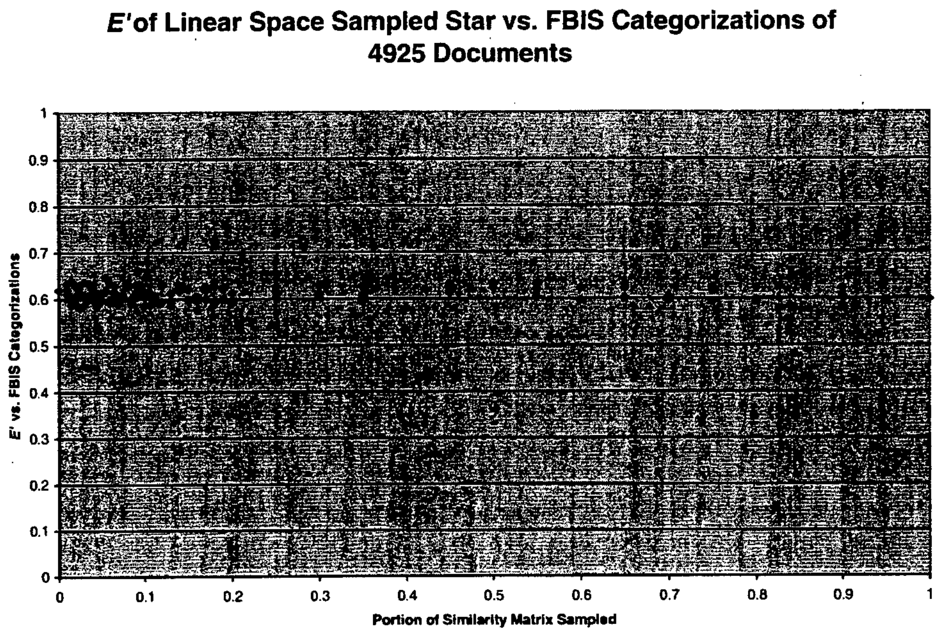


Figure 7: The effect of sampling on the quality of the clustering for the FBIS collection. The x-axis show the sampling size. The y axis shows the E-measure computed relative to the human clustering.

E-measure for 32 machines is about 41 %.

5 Conclusion

We presented a scalable algorithm for information organization. Scalability is a very important property for information organization algorithms especially when the collections are dynamic and Web-based. We implemented these algorithms as a scalable system for information organization. In the near future, we plan to expand our experimental collection to demonstrate the performance of our algorithms when dealing with hundreds of thousands of documents.

References

Aslam, J., K. Pelekhov, and D. Rus. (1998). Static and Dynamic Information Organization with Star Clusters. In *Proceedings of the 1998 Conference on Information Knowledge Management*, Baltimore, MD.

Aslam, J., K. Pelekhov, and D. Rus. (1999). A practical Clustering Algorithm for Static and Dynamic Information Organization. In *Proceedings of the 1999 Symposium on Discrete Algorithms*, Baltimore, MD.

Charikar, M., C. Chekuri, T. Feder, and R. Motwani. (1997). Incremental clustering and dynamic information retrieval, in *Proceedings of the* 29^{th} *Symposium on Theory of Computing.*

Croft, W.B. A model of cluster searching based on classification. *Information Systems*, 5:189-195, 1980.

Cutting, D., D. Karger, and J. Pedersen. (1993). Constant interaction-time scatter/gather browsing of very large document collections. In *Proceedings of the* 16^{th} *SIGIR*.

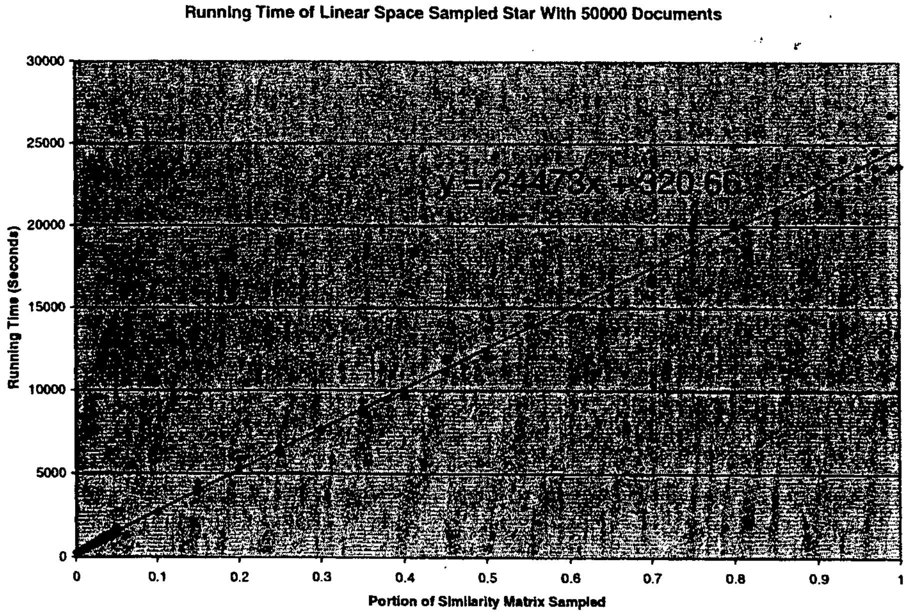


Figure 8: The running time of the Linear Sampled Star algorithm on a 50000 document subset of TREC. The x-axis shows the sample size and the y axis shows the running time in seconds.

Everitt, B. (1993). *Cluster Analysis.* Arnold, London.

Hearst, M. and J. Pedersen. (1996). Reexamining the cluster hypothesis: Scatter/Gather on Retrieval Results. In *Proceedings of the* 19^{th} *SIGIR.*

Jain, A. and R. Dubes. (1988). *Algorithms for Clustering Data*, Prentice Hall.

Jardine, N. and C.J. van Rijsbergen. (1971). The use of hierarchical clustering in information retrieval, 7:217-240.

Mirkin, B. (1996). *Mathematical classification and clustering.* Kluwer Academic Publishers, Boston.

van Rijsbergen, C.J.. (1979). *Information Retrieval.* Butterworths, London.

Rus, D., R. Gray, and D. Kotz. (1997). Transportable Information Agents. *Journal of Intelligent Information Systems*, vol 9. pp 215-238.

Sibson, P. (1973). SLINK: an optimally efficient algorithm for the single link cluster method. *Computer Journal* 16, pp30–34.

Silverstein, C. and J. Pedersen. (1997) Almost-Constant-Time Clustering of Arbitrary Corpus Subsets. In *Proceedings of SIGIR*, pp 60–66.

Voorhees, E. (1985). The cluster hypothesis revisited. In *Proceedings of the* 8^{th} *SIGIR*, pp 95-104.

Willett, P. (1988). Recent trends in hierarchical document clustering: A critical review. *Information Processing and Management*, 24:(5):577-597.

Worona, S. (1971). Query clustering in a large document space. In Ed. G. Salton, *The SMART Retrieval System*, pp 298-310. Prentice-Hall.

Zuckerman, D. (1993). NP-complete problems have a version that's hard to approximate. In *Proceedings of the Eight Annual Structure in Complexity Theory Conference*, IEEE Computer Society, 305–312, 1993.

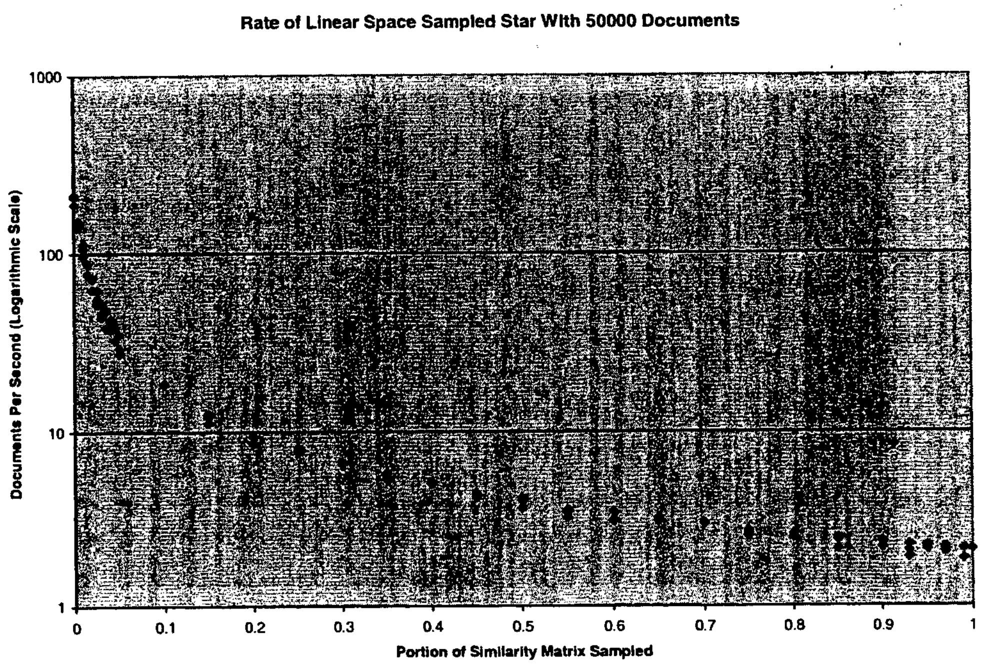


Figure 9: The effect of the sample size on the rate of the Linear Space Sampled Star algorithm (plotted on a logarithmic scale).

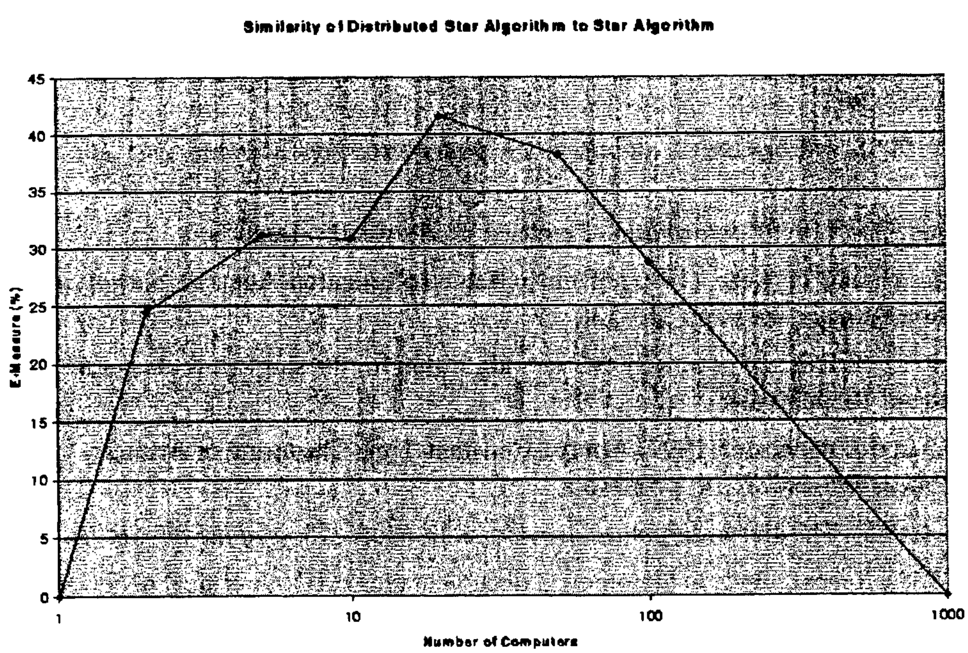


Figure 10: This graph shows the E-measure of the distributed star algorithm relative to the off-line star clustering of the same document set.

Mobile Information Agents

Daniela Rus
Dartmouth College
Hanover, New Hampshire

February 12, 1999

Acknowledgements

Map data

Geographic Data Technology

GDT
GEOGRAPHIC DATA TECHNOLOGY

Summer interns

- Michael Bahl, Colorado State
- Alex Roetter, Stanford

Demo team

- Katya Pelekhov
- Mark Montague
- Robert Grube
- Daniel Bilar
- Ron Peterson
- Alik Widge
- Jon Howell
- Clint Hepner
- Alex Iliev
- Guanling Chen
- Rong Xie
- Jon Bredin
- Debbie Greenberger
- Brian Brewington
- Katsuhiro Moizumi
- Shankar Sundaram
- Guofei Jiang
- Ken Yasuhara

Our Vision

- Pervasive mobile-agent systems
- Dynamic Information Sources

Educational Applications

Commercial Applications

Military Applications

Technology components

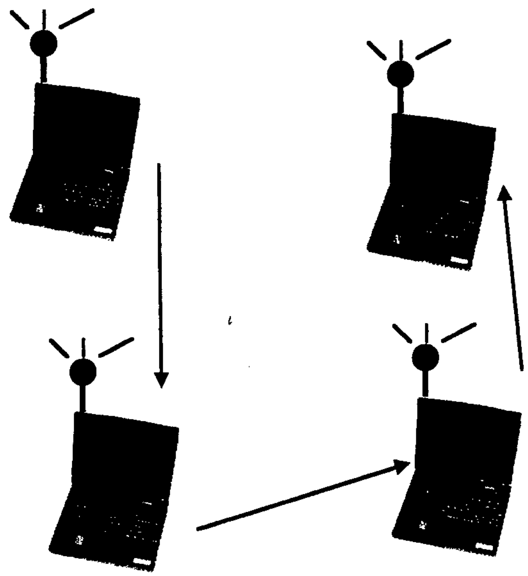
Active software → Mobile agents

Active information → Information clustering

Active networking → Wireless routing

Applications → Information organization; Topic detect. And track.; Military scenario

Mobile agents



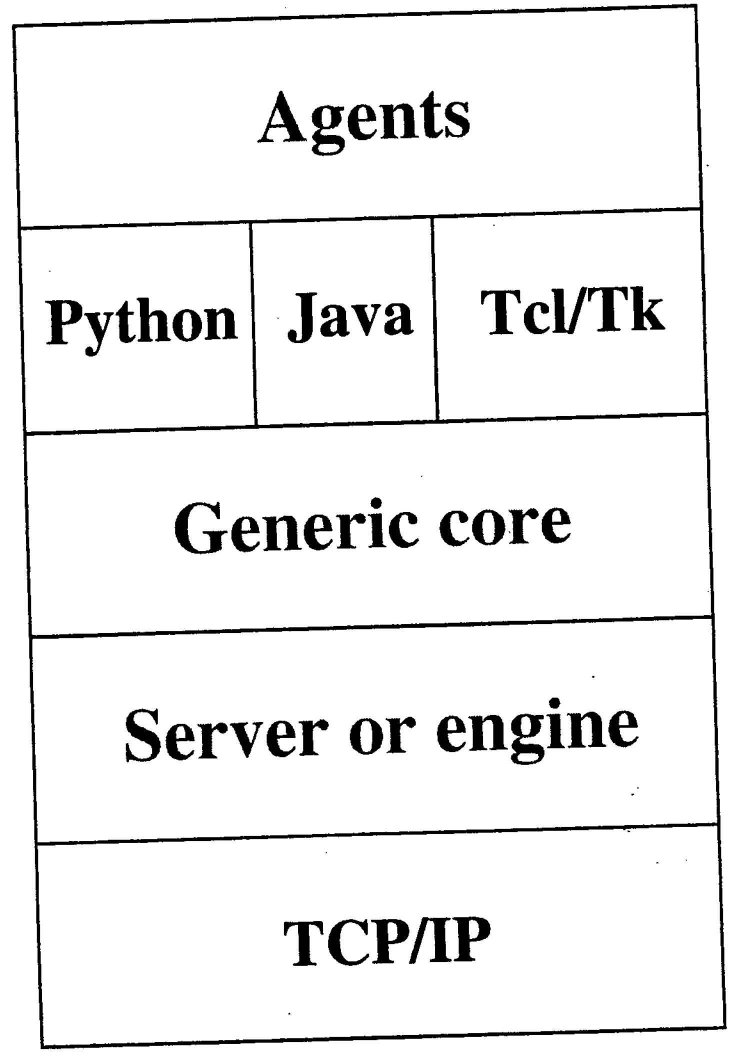
- Executing program
- Migrates <u>by choice</u>
- Heterogeneous systems
- Interacts with other agents

- Efficient
  - Access resources locally
  - Offload from underpowered client
  - Offload from overloaded servers
- Mobile computing
- Dynamic deployment
- Convenience

Single general framework in which distributed applications can be implemented easily, efficiently and robustly

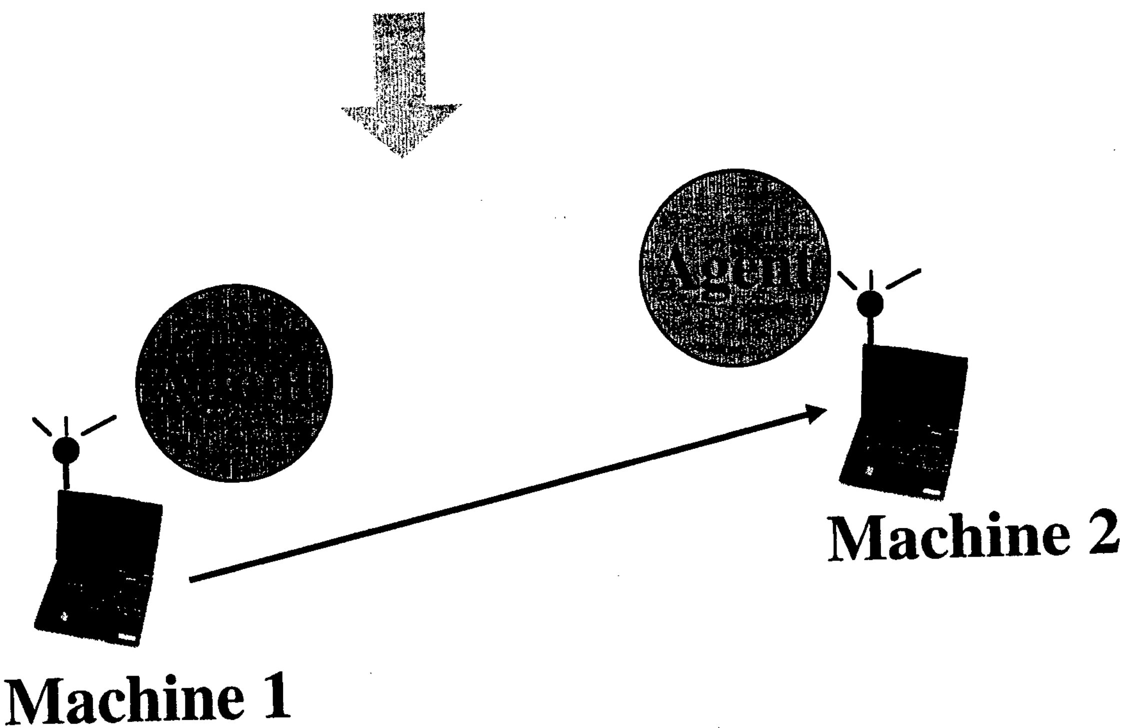
D'Agents (Dartmouth)

- Languages
  - Tcl, Java and Scheme
  - Python in progress
- Performance
  - Halfway there
- Support services
- Security
  - Machine protection
  - Agent protection while <u>in transit</u>
  - No agent protection while <u>on a machine</u>



Basic Agent Capabilities

- Agent_Jump
- Messages
- Agent Meet
- Fork, Clone, etc.



System Architecture

Agent Apps
Agents
Tcl/Tk, Java
Generic Core
Server
TCP/IP, email

→

Core Stubs + Security

Agent Navigation

- Virtual Sensors or the state of the network and software changes
- Yellow Pages for locating services
- Navigation Plans

Information Capture &Access

- Agents travel to distributed information sources
- Agents can operate disconnected
- Agents can process and integrate large amounts of data
- Agents can monitor data sources
- Agents can filter information in a customized fashion

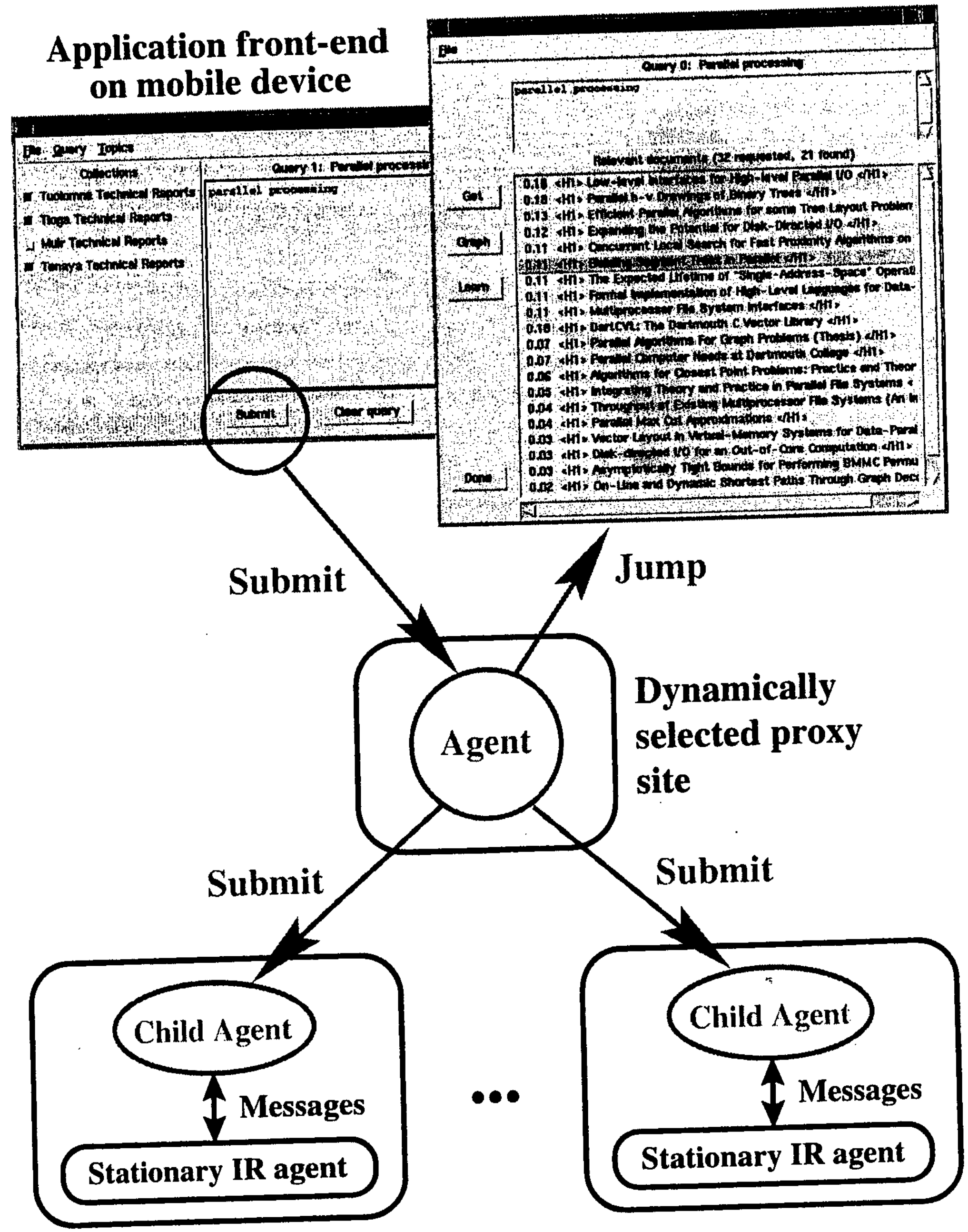
Mobile agents for IR

Send the search code to the location of the data, rather than downloading the data

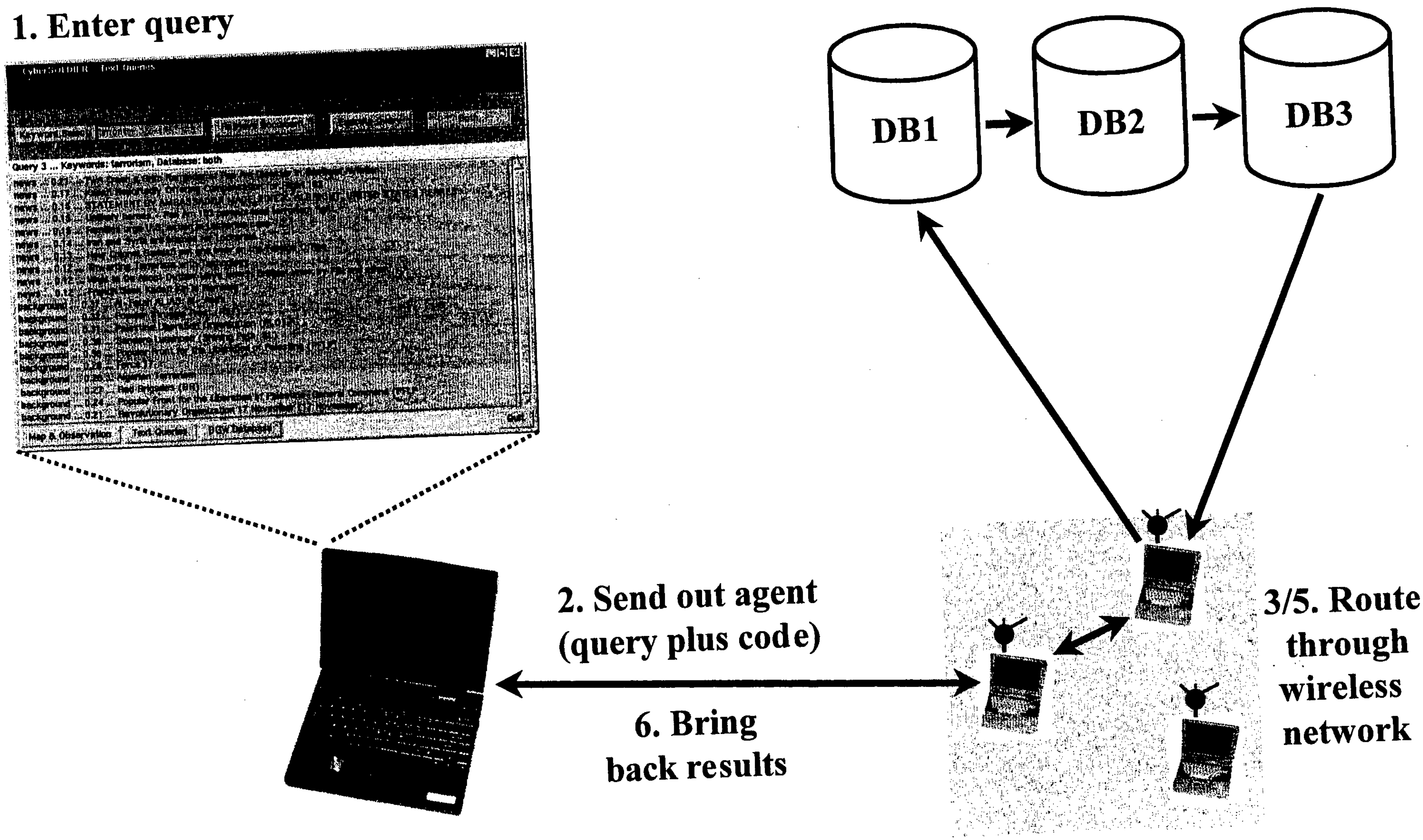
- Reduce end-to-end latency
- Conserve bandwidth
- Continue even when disconnected

EVEN IF

- Information source has a low-level interface



Details


1. Enter query
Query 3 ... Keywords: terrorism, Database: both
4. Visit databases / merge and filter results
DB1
DB2
DB3
2. Send out agent (query plus code)
6. Bring back results
3/5. Route through wireless network

Information Processing

- AGENTS:
- Can handle lots of data
- Can structure lots of data
- Can summarize lots of data
- Do not have situation knowledge

- HUMANS:
- Hard to deal with information overload
- Hard to find structure in lots of data
- Have situation knowledge
- Make decisions

Information Organization

Information Base (Raw Data) → Knowledge Base (Organized Data)

Information Organization Agents
Star Cluster System

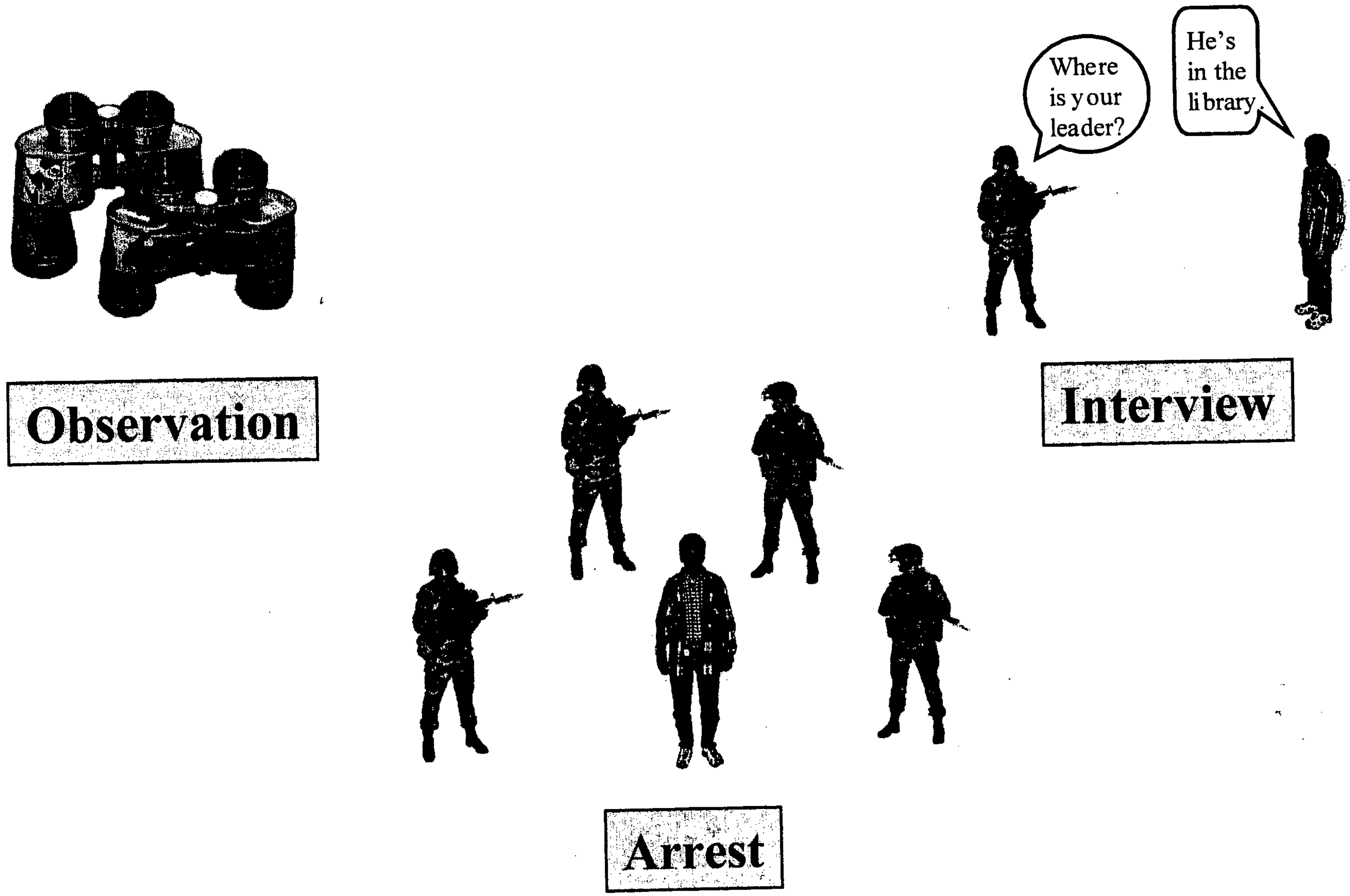
Documents $\xrightarrow{\text{Indexing}}$ **Information bases** $\xrightarrow{\text{Clustering}}$ **Knowledge bases**

Application: The plot

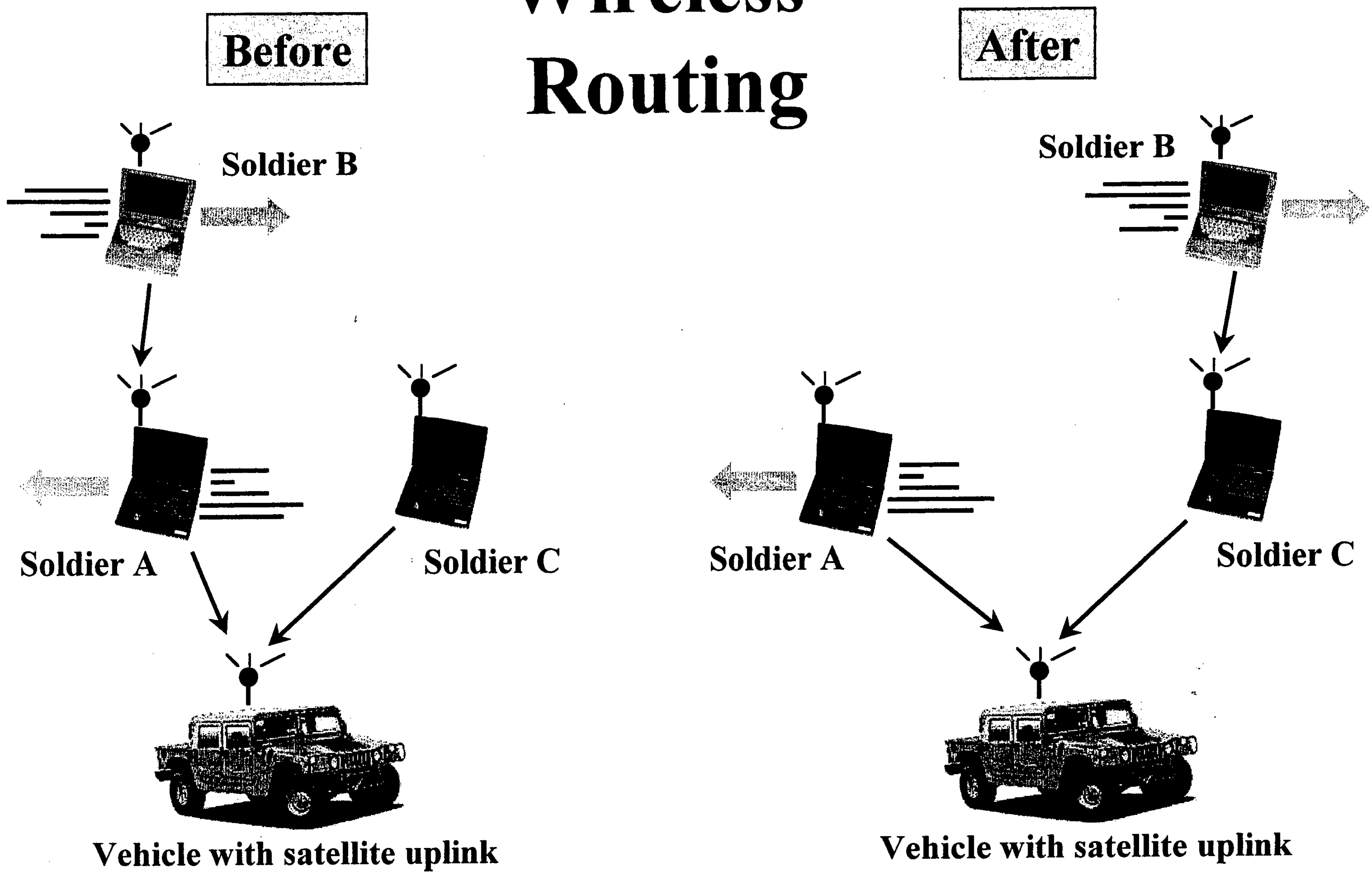
Hanover, New Hampshire

- Hanover: oldest city in an imaginary country
- First democratic elections in 30 years
- U.S. military in-country to oversee elections
- Terrorist group in-country to disrupt elections
- Terrorist’s most likely target: Hanover
- Mission: identify, stop and arrest the terrorists

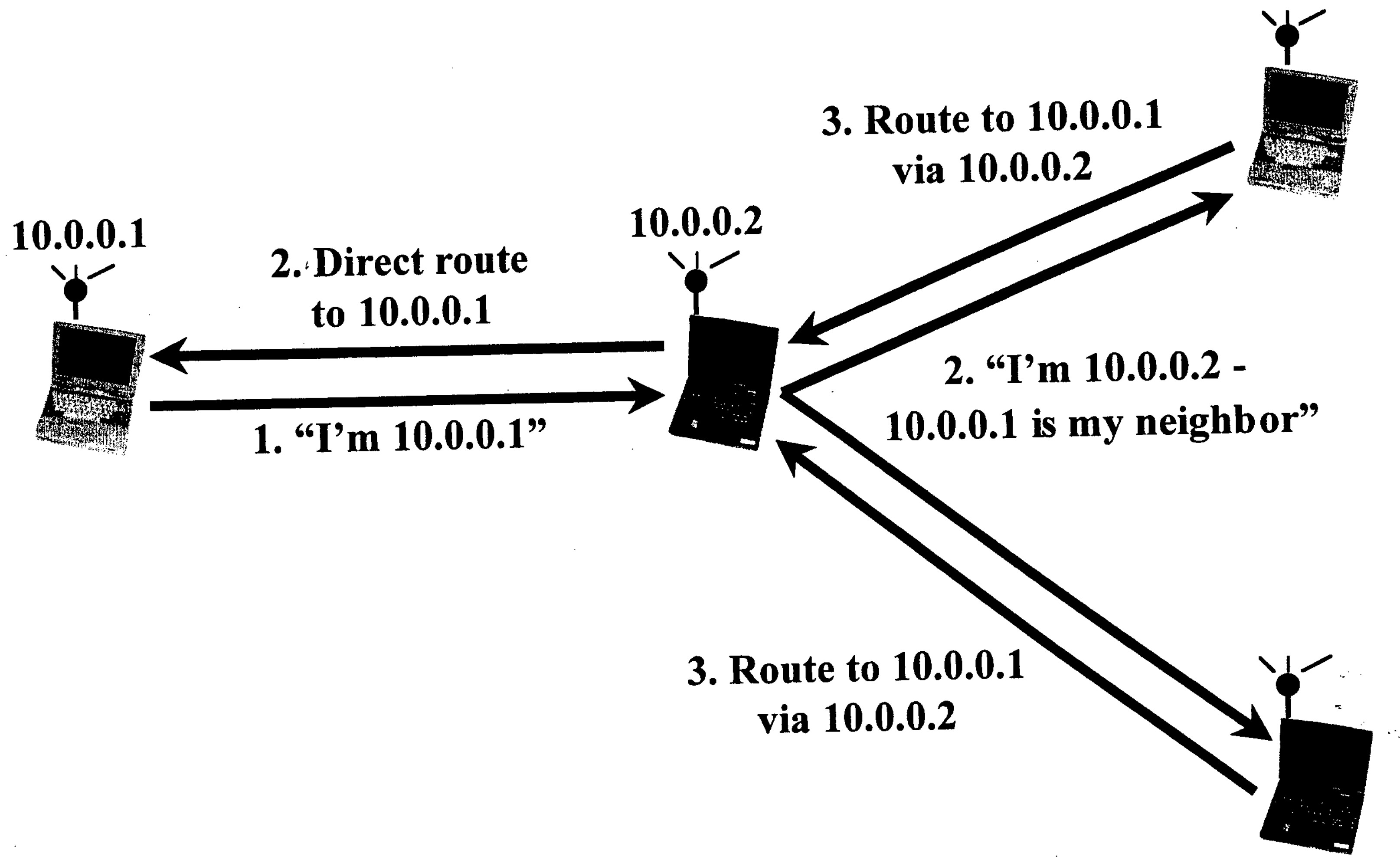
Three mission phases



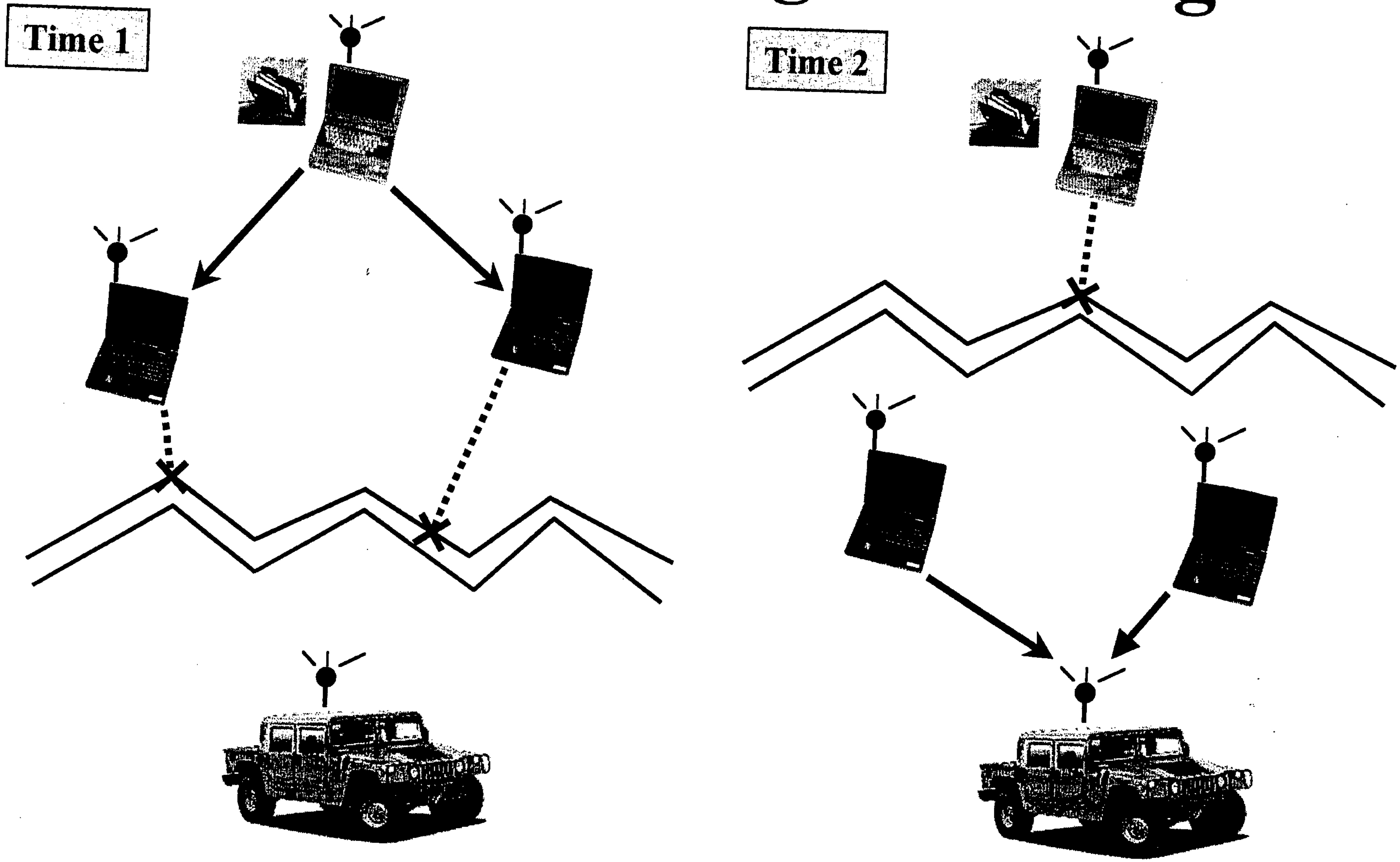
Wireless Routing



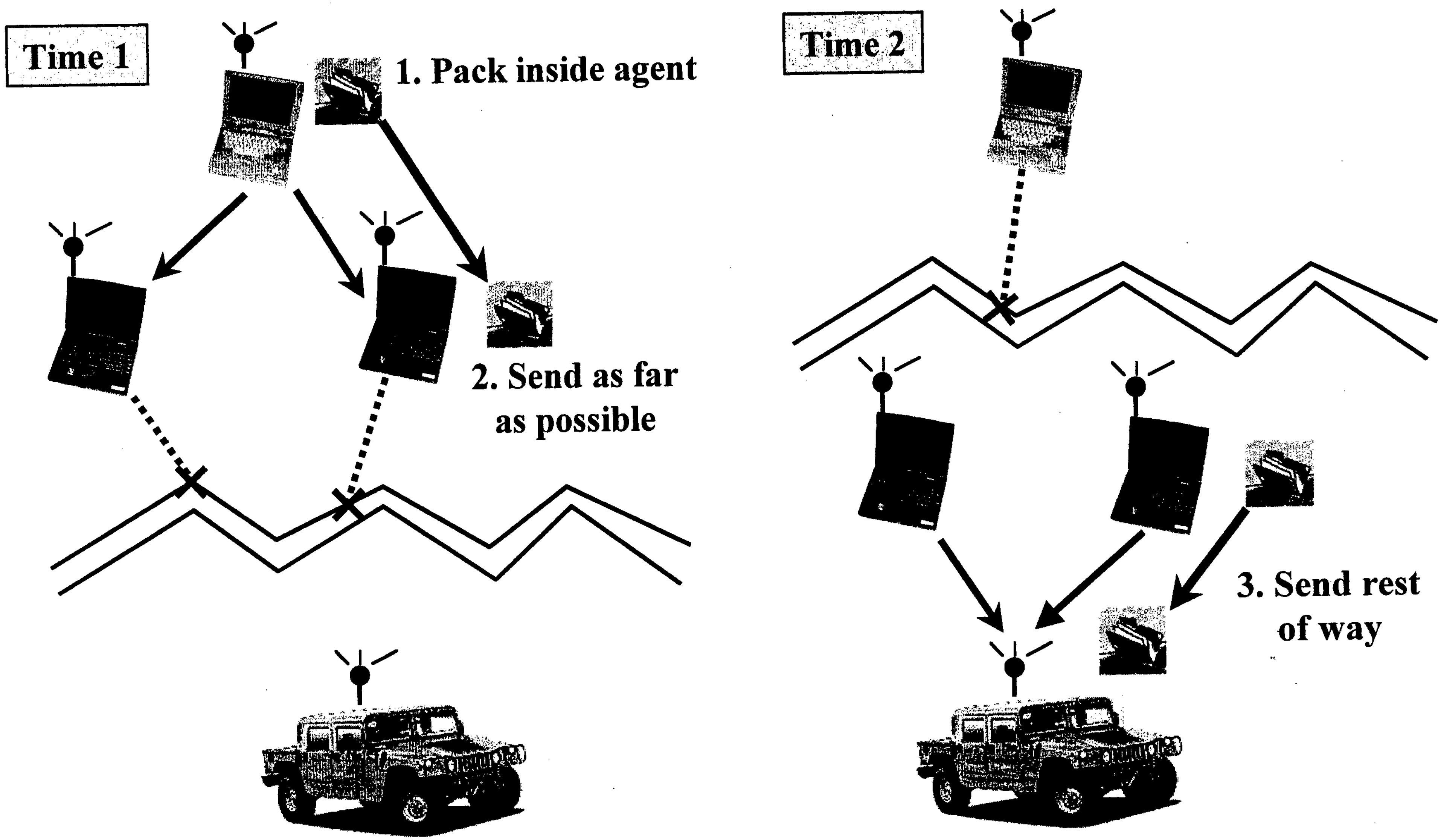
APRL (Harvard)



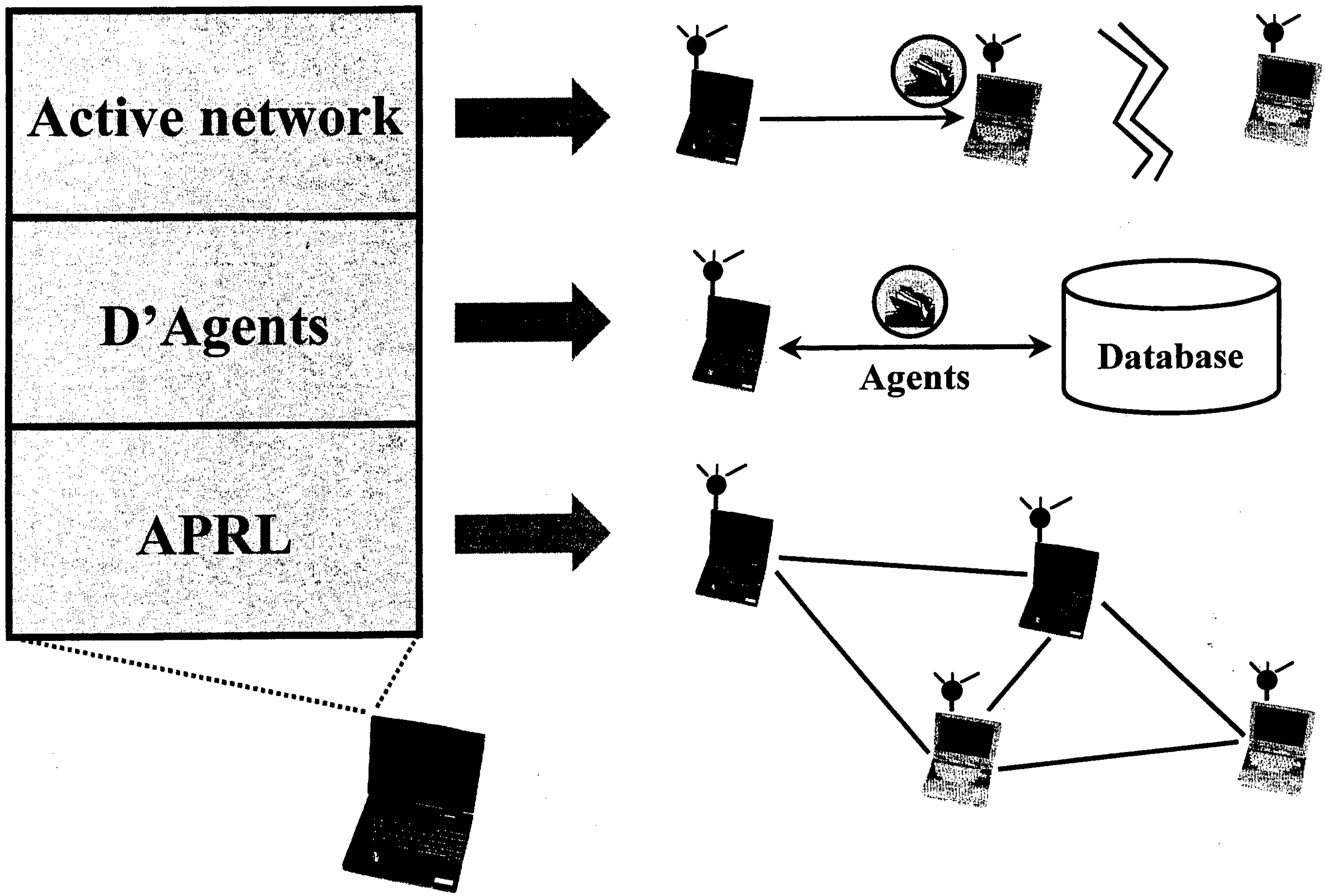
Wireless routing not enough



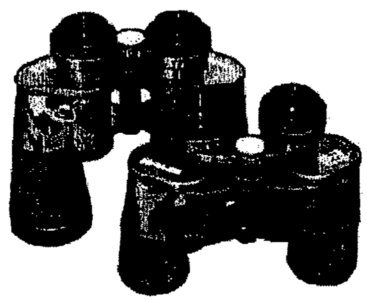
Active networking prototype



Demo architecture



Phase one: Vehicle observation



The players

- 4 stationary observers
- 2 "suspicious" cars

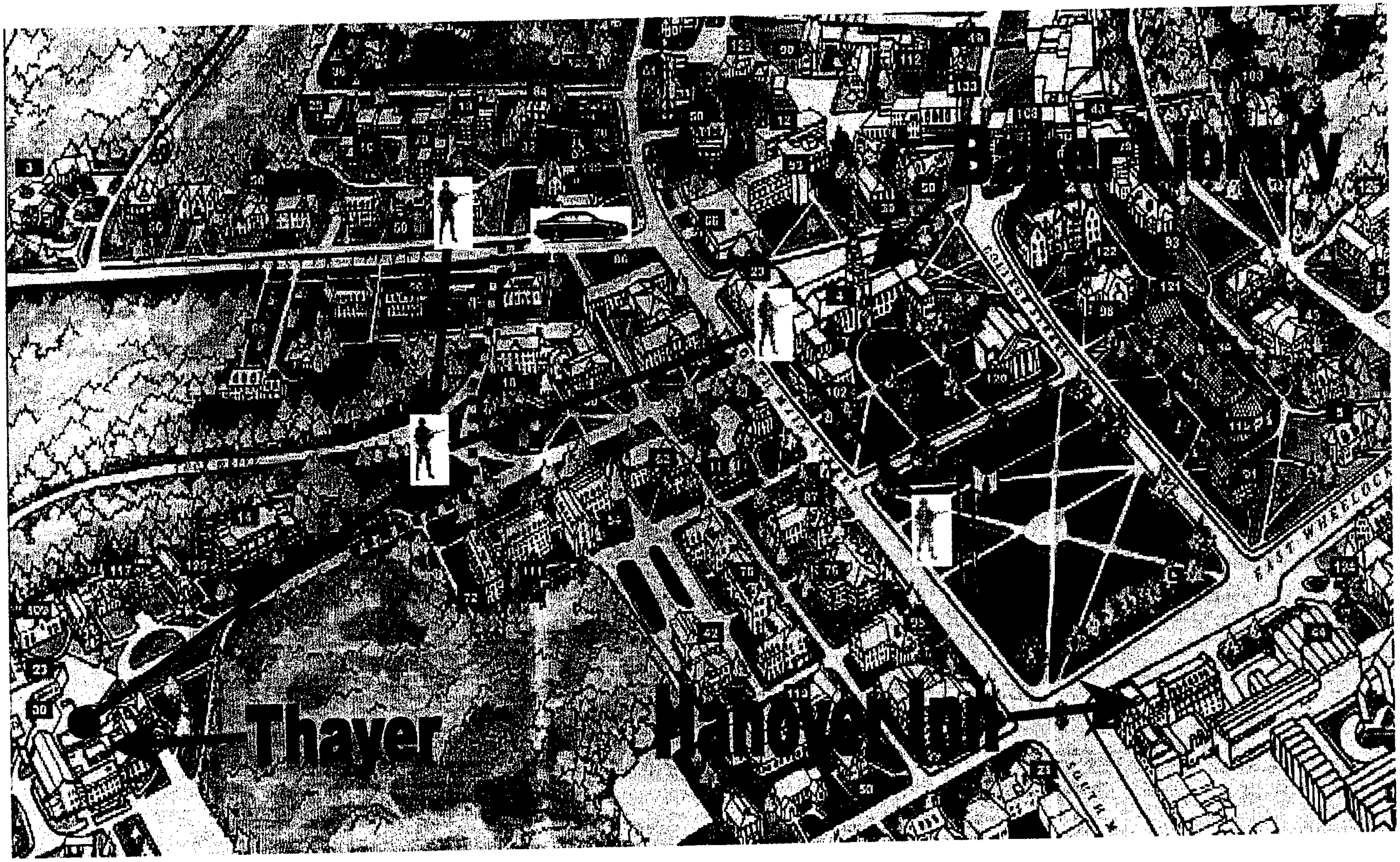
The plot

- Observers report the current positions of the cars

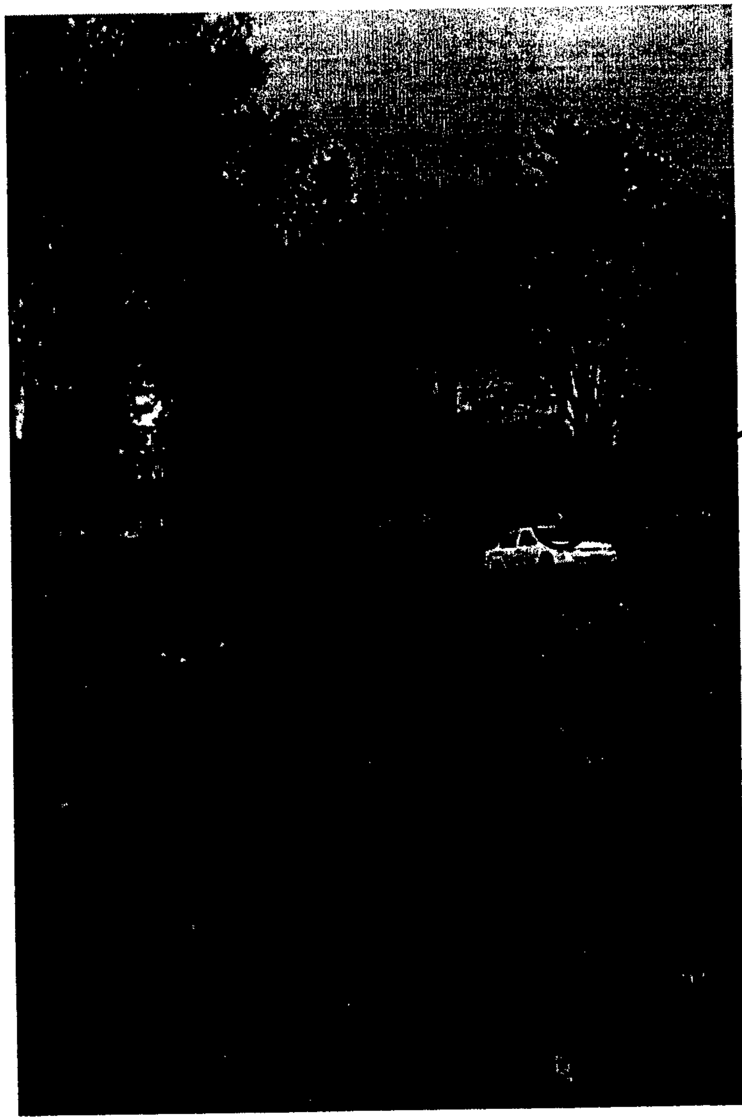
Demonstrates

- APRL
- Active network

Physical Layout

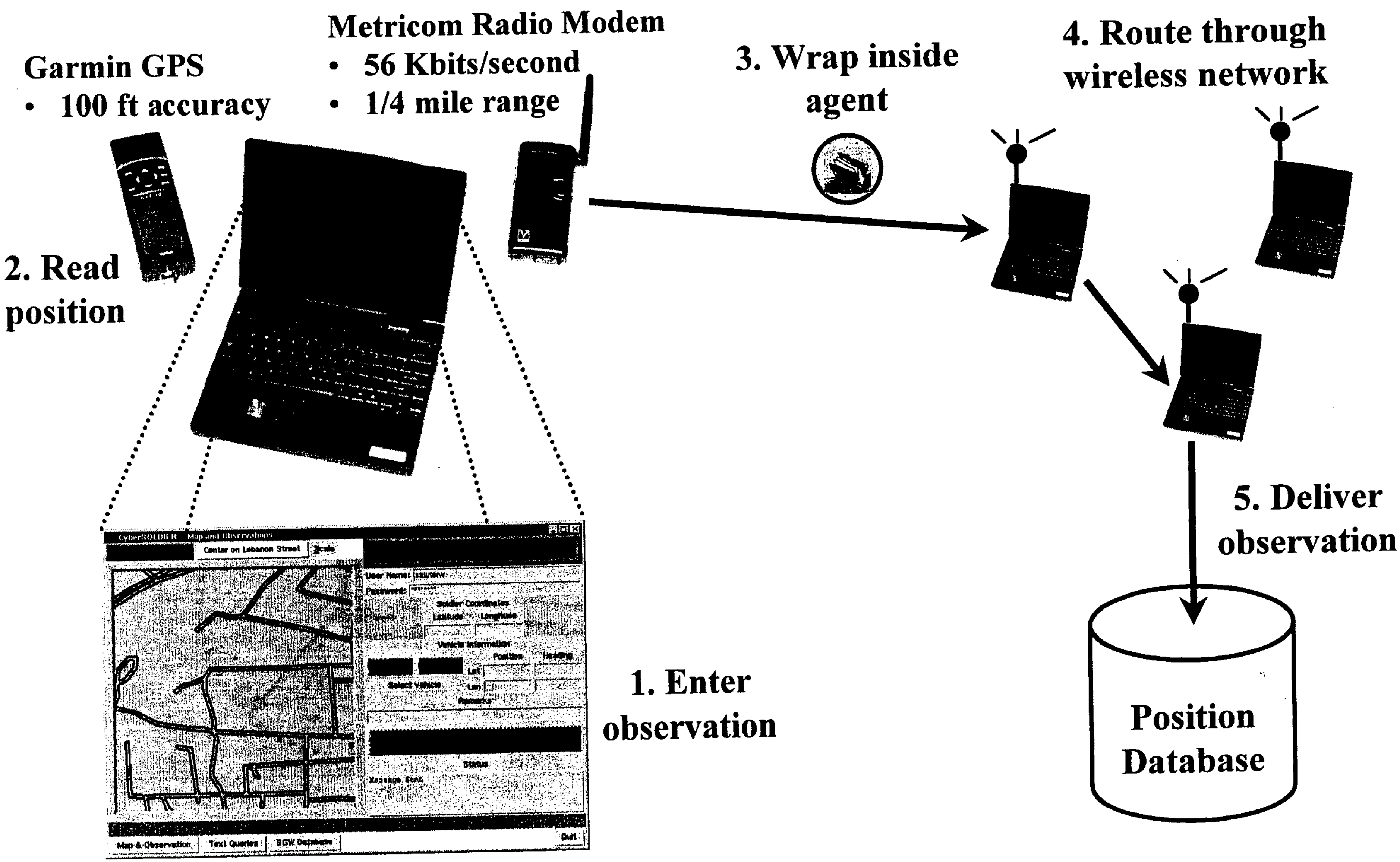


An observer observing



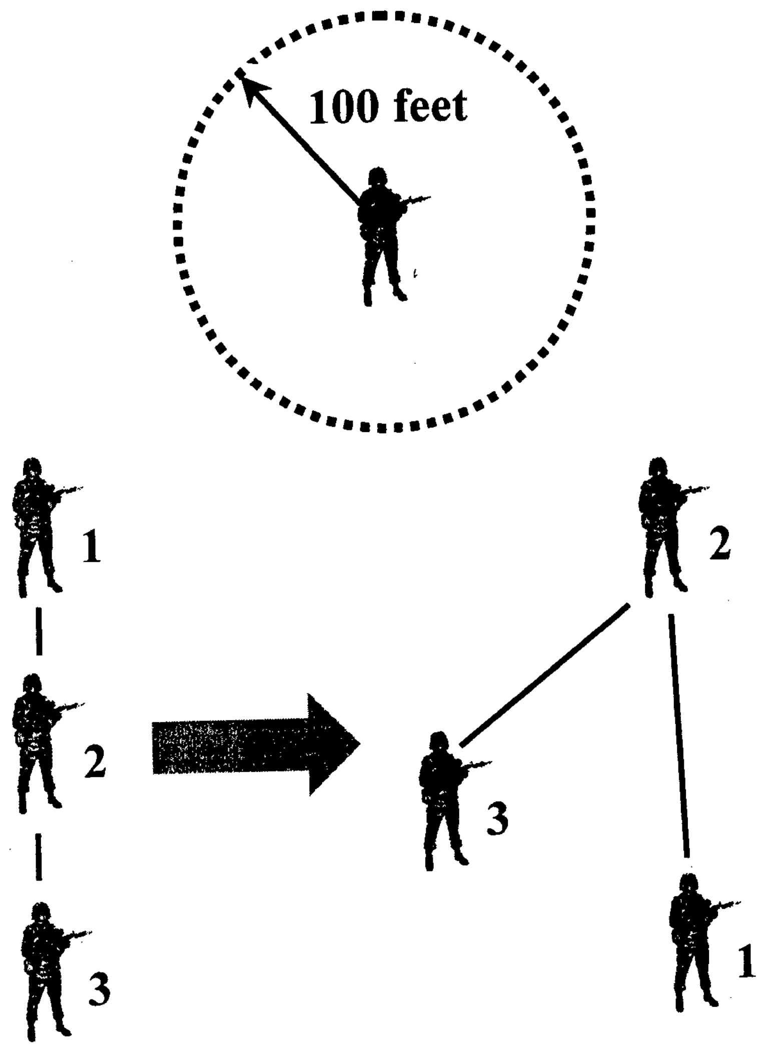
Suspicious car

Details

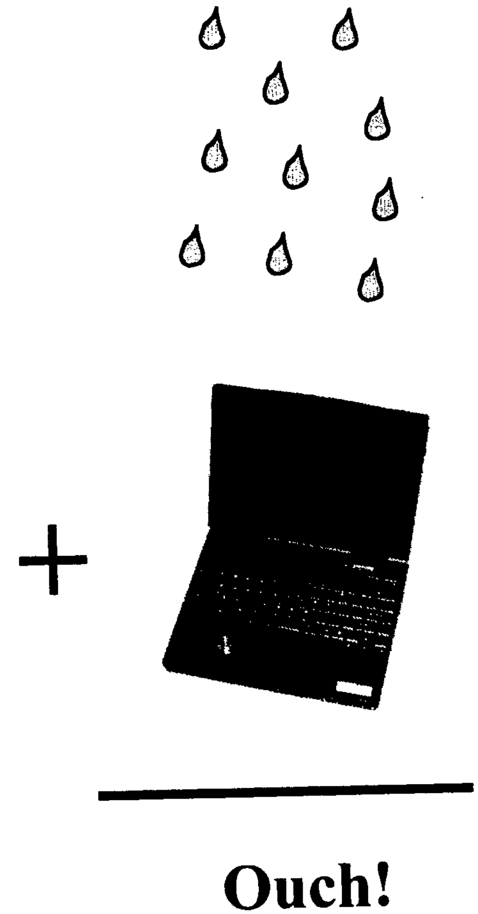

Garmin GPS
• 100 ft accuracy
Metricom Radio Modem
• 56 Kbits/second
• 1/4 mile range
2. Read position
3. Wrap inside agent
4. Route through wireless network
1. Enter observation
5. Deliver observation
Position Database

Notes

GPS errors



Rain



Future work

Scalability

- More agents per machine
  - Multi-threaded agent server
  - "Hot" interpreters
  - Faster interpreters
- More agents in the network
  - Compression
  - Code caching
- More machines per cloud
  - New versions of APRL
  - New routing algorithms

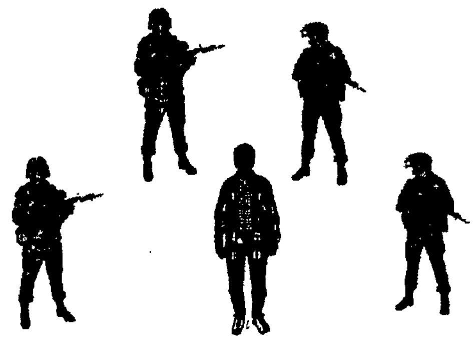
Active networking

- Lower-level implementation
- Different routing algorithms

Hypothesis tracker

Automatically determine which observations correspond to the same vehicle

Phase two: Arrest



The players

- 5 moving soldiers
- 1 stationary soldier
- 1 suspected terrorist

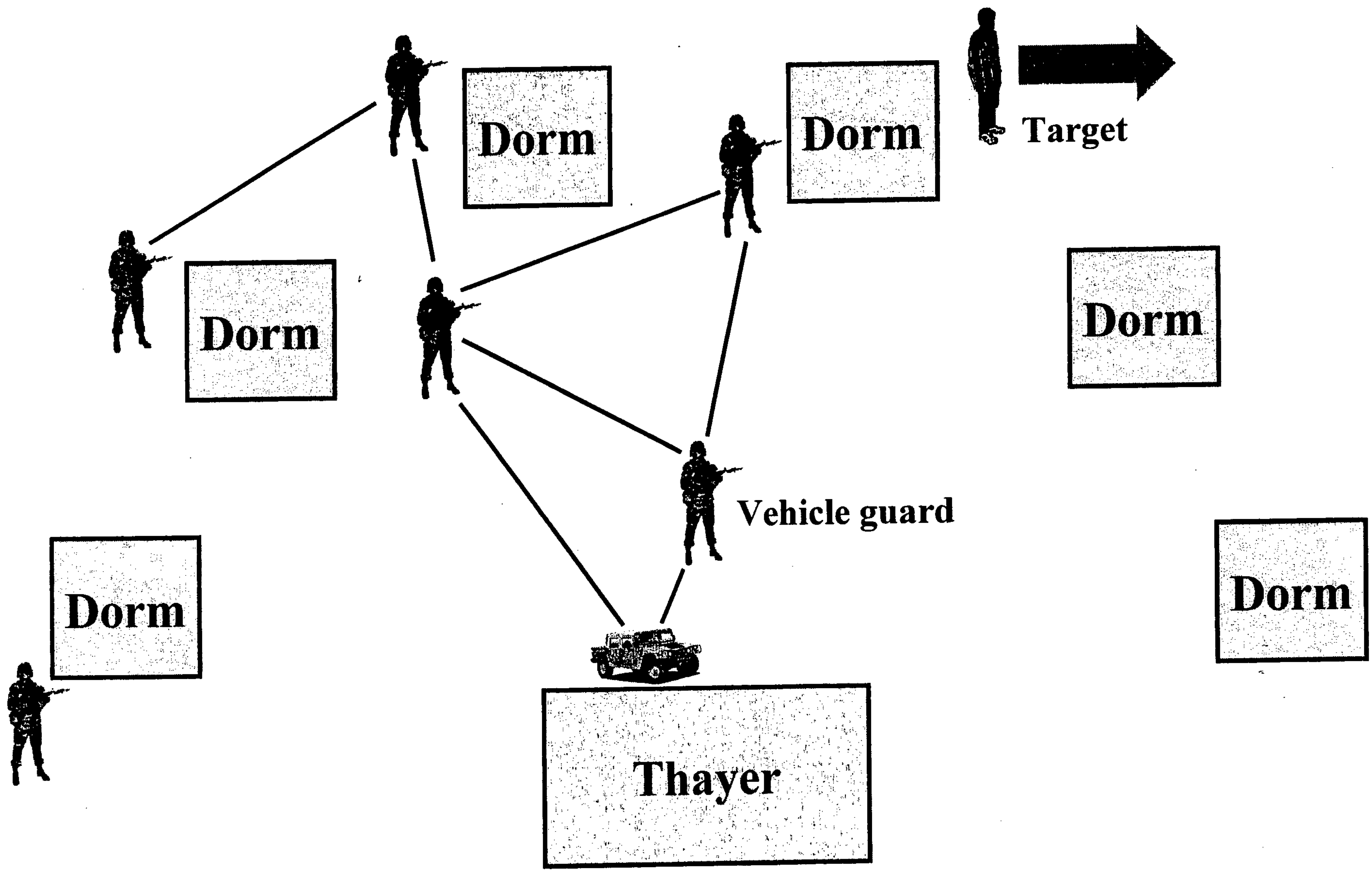
The plot

- Soldiers follow the suspect
- Soldiers' laptops report their position

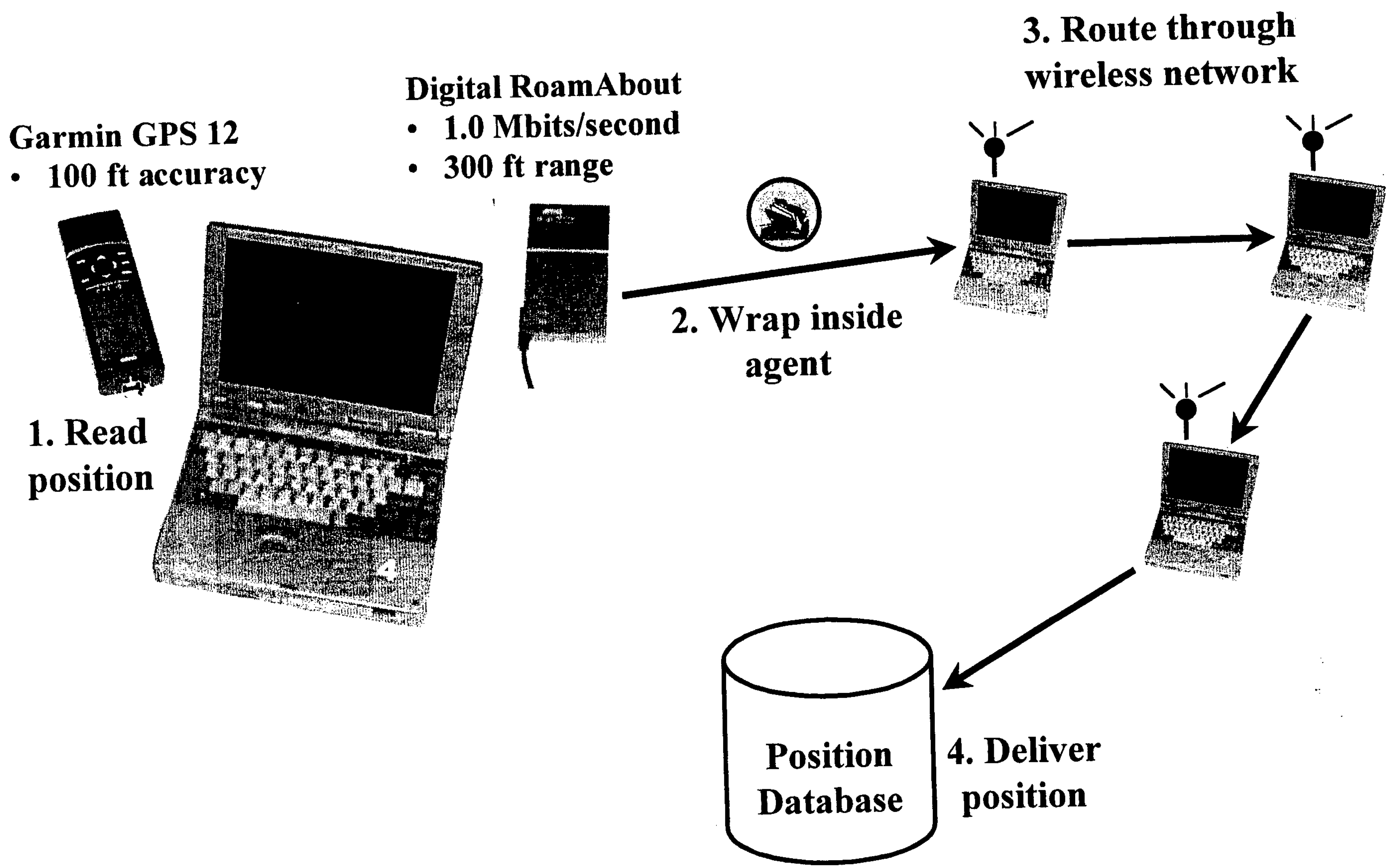
Demonstrates

- APRL
- Active network

Physical layout



Details


Garmin GPS 12
• 100 ft accuracy
Digital RoamAbout
• 1.0 Mbits/second
• 300 ft range
1. Read position
2. Wrap inside agent
3. Route through wireless network
Position Database
4. Deliver position

Future work

Scalability

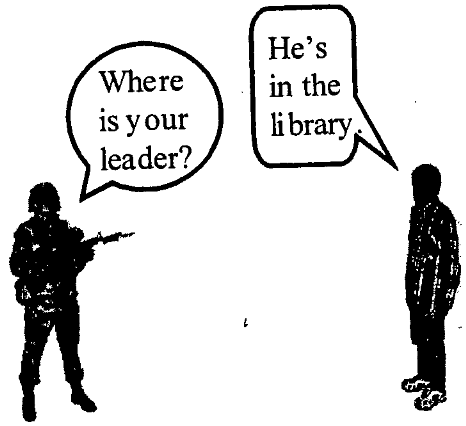
Active networking

Differences vs. vehicle observation

- Short-range transmitters
- Moving objects

- Network must adapt faster.
- Routing software must send out more probe packets.

Phase three: Interview



The players

- 1 interviewer
- 1 townsperson or suspect

The plot

- Interviewer runs queries against various databases

Demonstrates

- D'Agents
- Information clustering

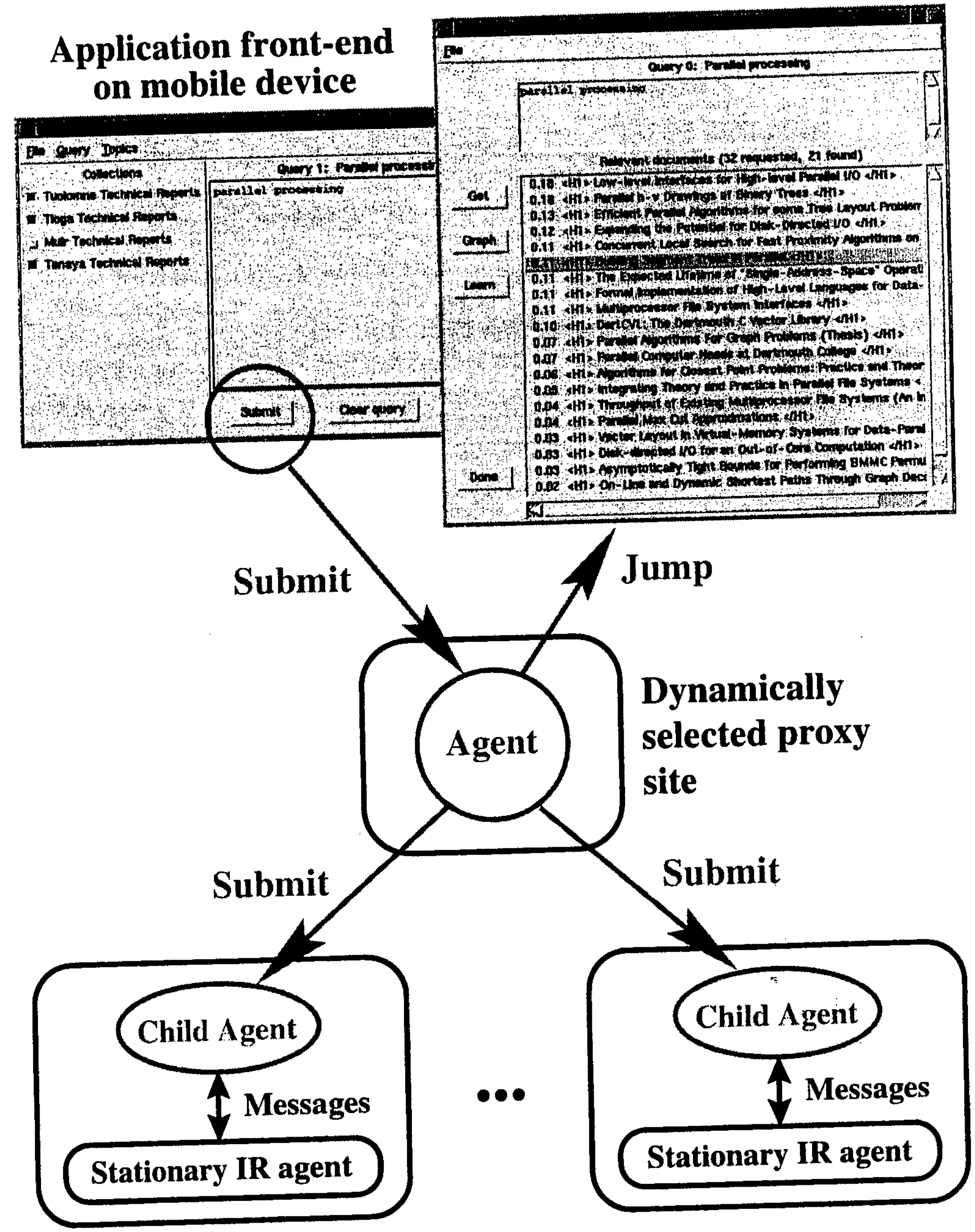
Mobile agents for IR

Send the search code to the location of the data, rather than downloading the data

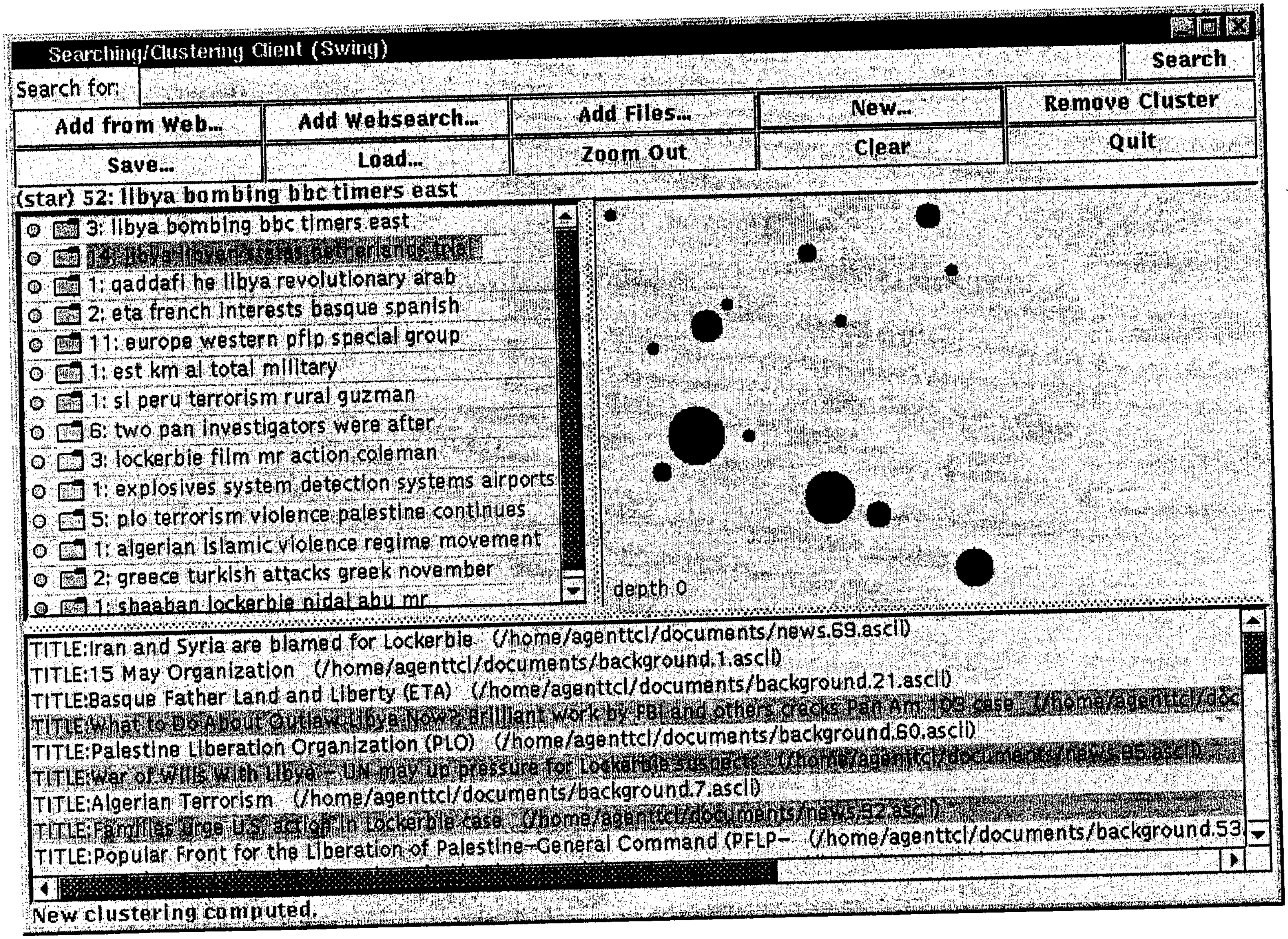
- Reduce end-to-end latency
- Conserve bandwidth
- Continue even when disconnected

EVEN IF

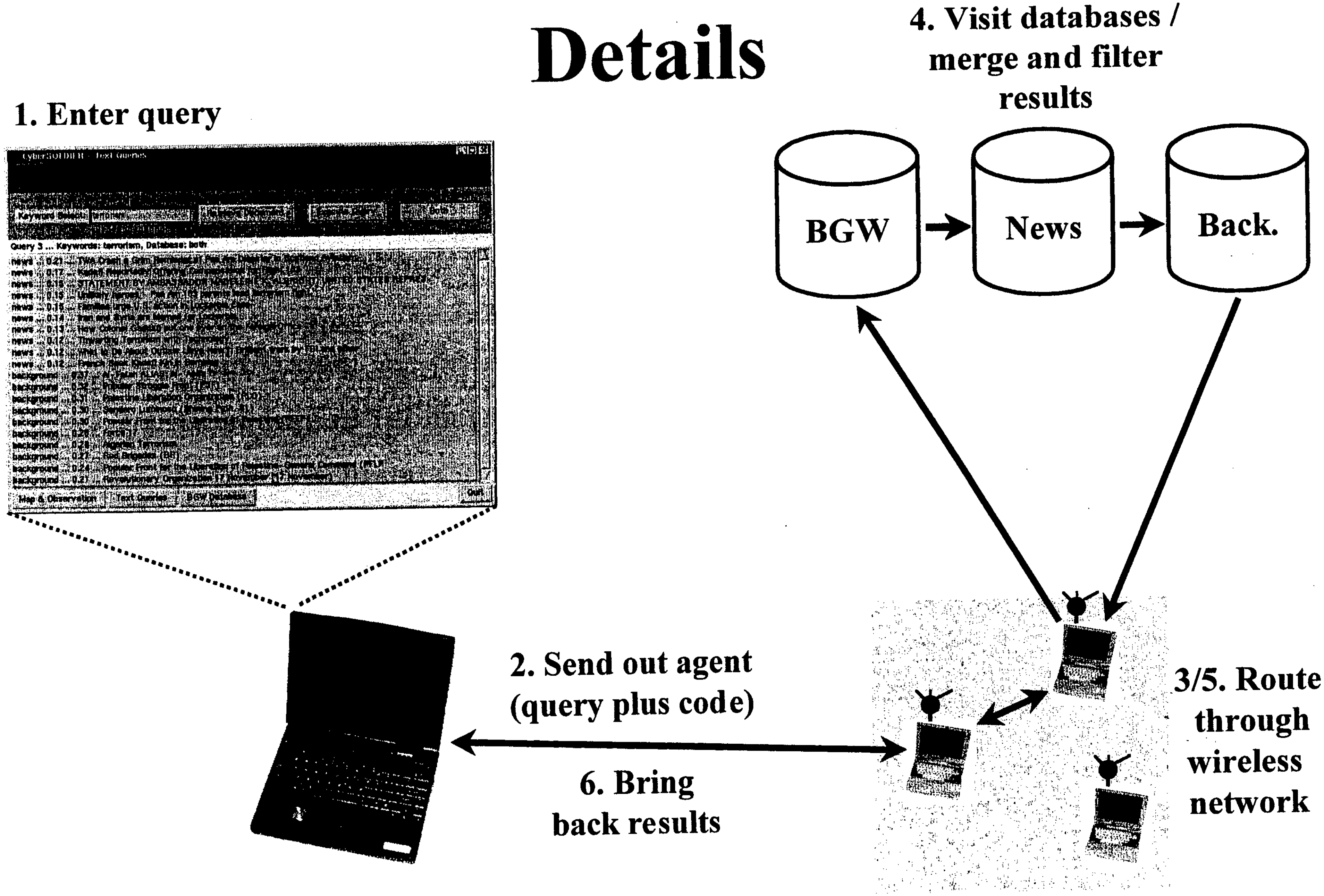
- Information source has a low-level interface



Document clustering



Details


4. Visit databases / merge and filter results
1. Enter query
Query 3 ... Keywords: terrorism, Database: both
BGW
News
Back.
2. Send out agent (query plus code)
6. Bring back results
3/5. Route through wireless network

Current and Future work

Scalability

- More agents per machine
  - Multi-threaded agent server
  - "Hot" interpreters
  - Faster interpreters
- More agents in the network
  - Compression
  - Code caching
- Less work per agent
  - Proxies that cache query results

Clustering

- Cluster "quality"
- Graphical representation
- Efficiency

Information Overload

Topic detect./tracking

Knowledge Bases

Conclusions

- APRL: Good for small to medium-sized wireless clouds
- D'Agents: The "right" infrastructure
- Clustering: Effective tool for summaries;
- Information Organization reduces information overload
- Future work
  - Scalability
  - Knowledge base applications
  - Other user interfaces
  - "Rest of project"

Information Organization Algorithms and Applications

Ph.D. Thesis Defense

Thesis Committee:

Daniela Rus (Dartmouth)
Javed Aslam (Dartmouth)
David Kotz (Dartmouth)
James Allan (UMass Amherst)

1/4/01

Outline

- **Motivation**
  information organization fights information overload
- **Clustering Algorithm**
  algorithm properties, design, evaluation
- **Applications**
  retrieval, filtering, persistent queries
- **Conclusions**
  ...and future work

1/4/01

Information overload

- **What is information overload?**

  "too much data, not enough information"

- **Why does it happen?**

  getting data is easy

  misunderstanding and zealous service

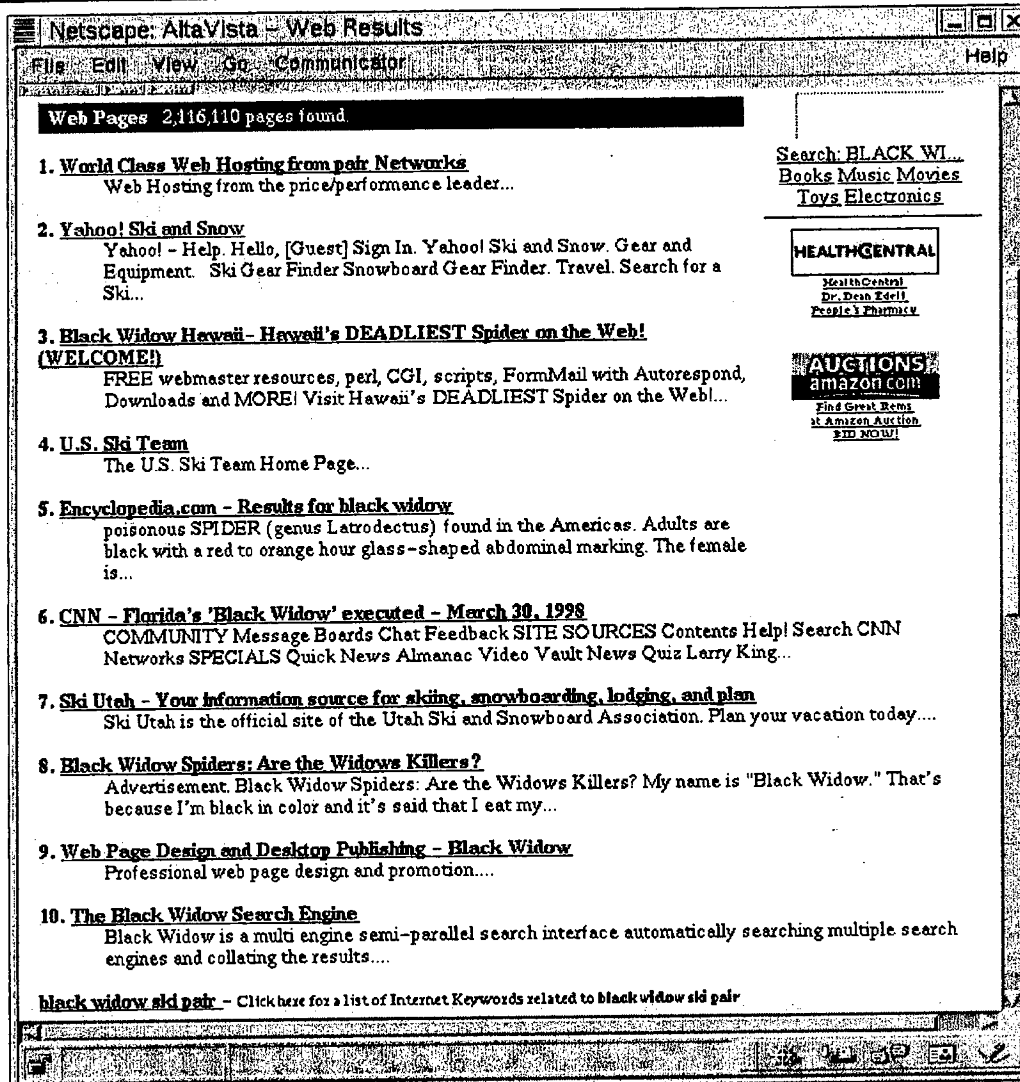
- **Example**

  search engines that generate ranked list of documents

  does not work if context is ambiguous

Misunderstood query

- **Query**
  "black widow ski pair"
- **Meaning**
  I am looking for skis called "black widows"
- **Search engine**
  spiders?
  au pair?
  ...?
  skiing?
- **Ranked list**
  all topics above



1/4/01

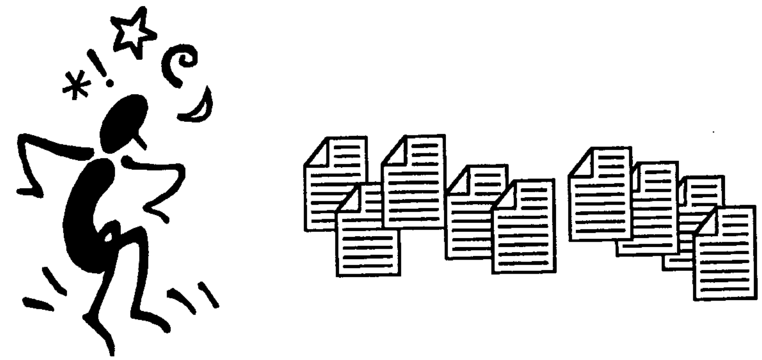
Information Organization

- **Prevailing approach**

  most commercial search engines

  find everything remotely relevant

  frustrated user



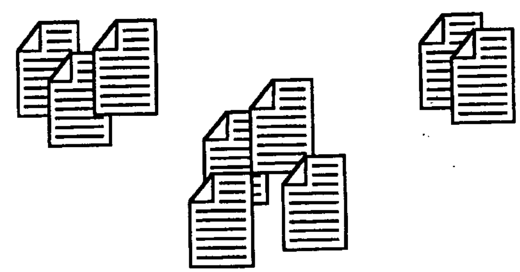
- **Organization approach**

  NorthernLight, many research systems

  organize retrieved document by topics

  manageable amount of information

  helps user to navigate and narrow the request



Other applications

- **Information overload is not limited to search**
- **Retrieval applications**

  Consider only text collections

  Classify by collection dynamics and user's interest

  Not consider nature of the documents (news events, stocks, etc.)

	searching	browsing	filtering
collection	stable	stable	changing
user's interests	changing	broad	stable

- **Hybrid applications**

  i.e., dynamic collection, changing user interests

  modern information system trend

Organizing system

- **Efficient**
- **Find accurate topics**
  topics that naturally occur in a collection
- **Update topics (add/delete documents)**
  while maintaining topic accuracy
- **Find topic hierarchies**
- **Compute topic summaries**
  readable (user)
  internal representation (programs)
- **Presentation**
  summarization and visualization of information

Methods for organization:
manual, classification, <u>clustering</u>

Related Work

- **Applications**
  - *Organizing retrieval results*
    - Hearst & Pedersen, 1996; NorthernLight; others
  - *Browsing*
    - Scatter/Gather, 1992; Yahoo!
  - *Filtering*
    - classification algorithms; Eichmann et al. 1998; others
  - *Other applications*
    - distributed retrieval, data mining, topic detection and tracking, ...
- **Methods**
  - *Cluster hypothesis*
    - similar documents are relevant to the same requests (van Rijsbergen 1971)
  - *Clustering algorithms*
    - single link, average link, k-means, others

Outline v2.0

- ~~Motivation~~
  ~~information organization fights information overload~~

  Desperately need an information organization system!

- Clustering Algorithm
  algorithm properties, design, evaluation

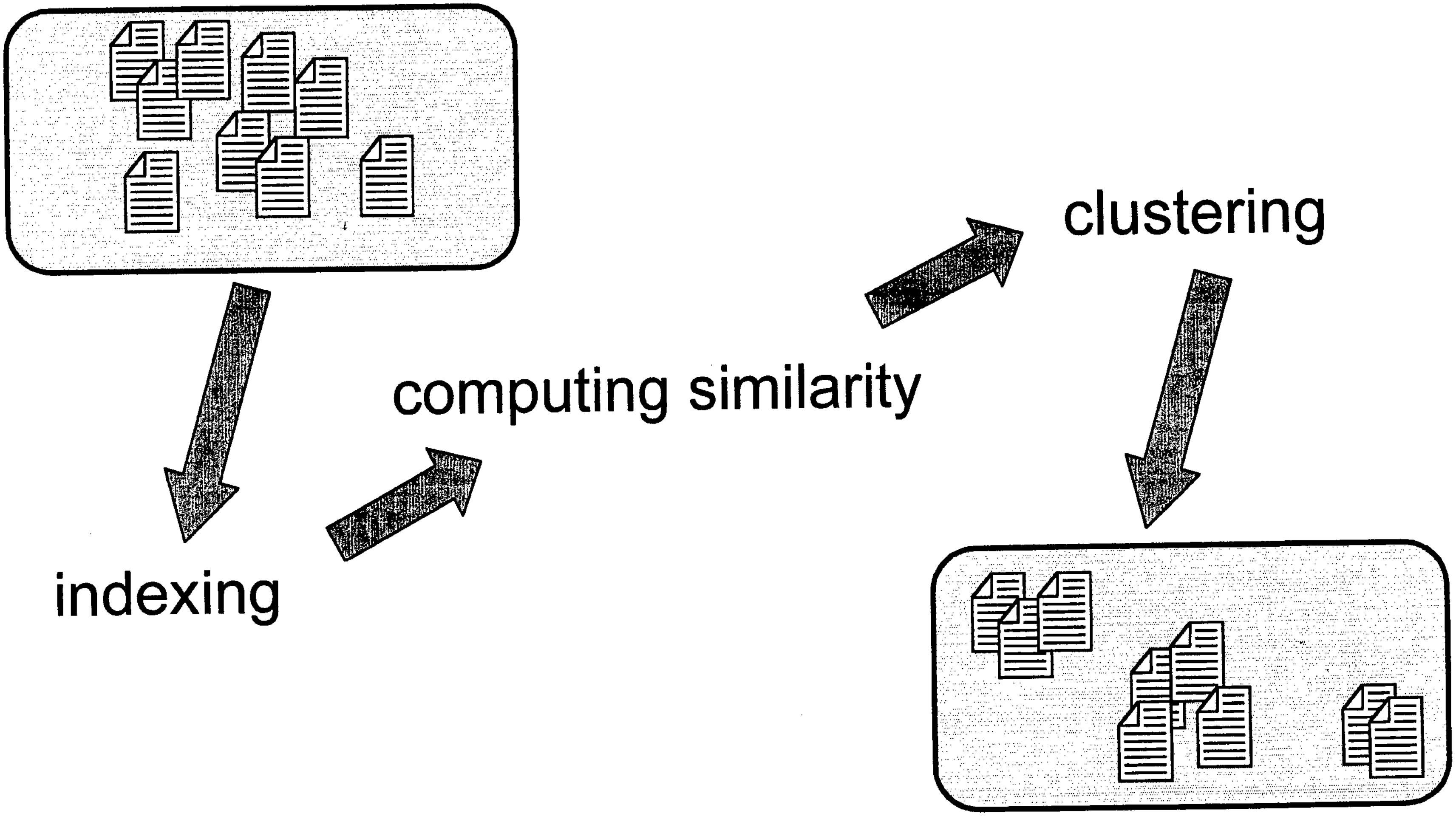
- Applications
  retrieval, filtering, other…

- Conclusions
  …and future work

Clustering Properties

	single link	average link	*K*-means	star
simple	P	P	P	P
efficient	P	P	P	P
accurate clusters		P		P
efficient updates	P			P
overlapping clusters			P	P
cluster hierarchy	P	P	P	P

Clustering system design



Vector Space model

- "Bag of words" document representation

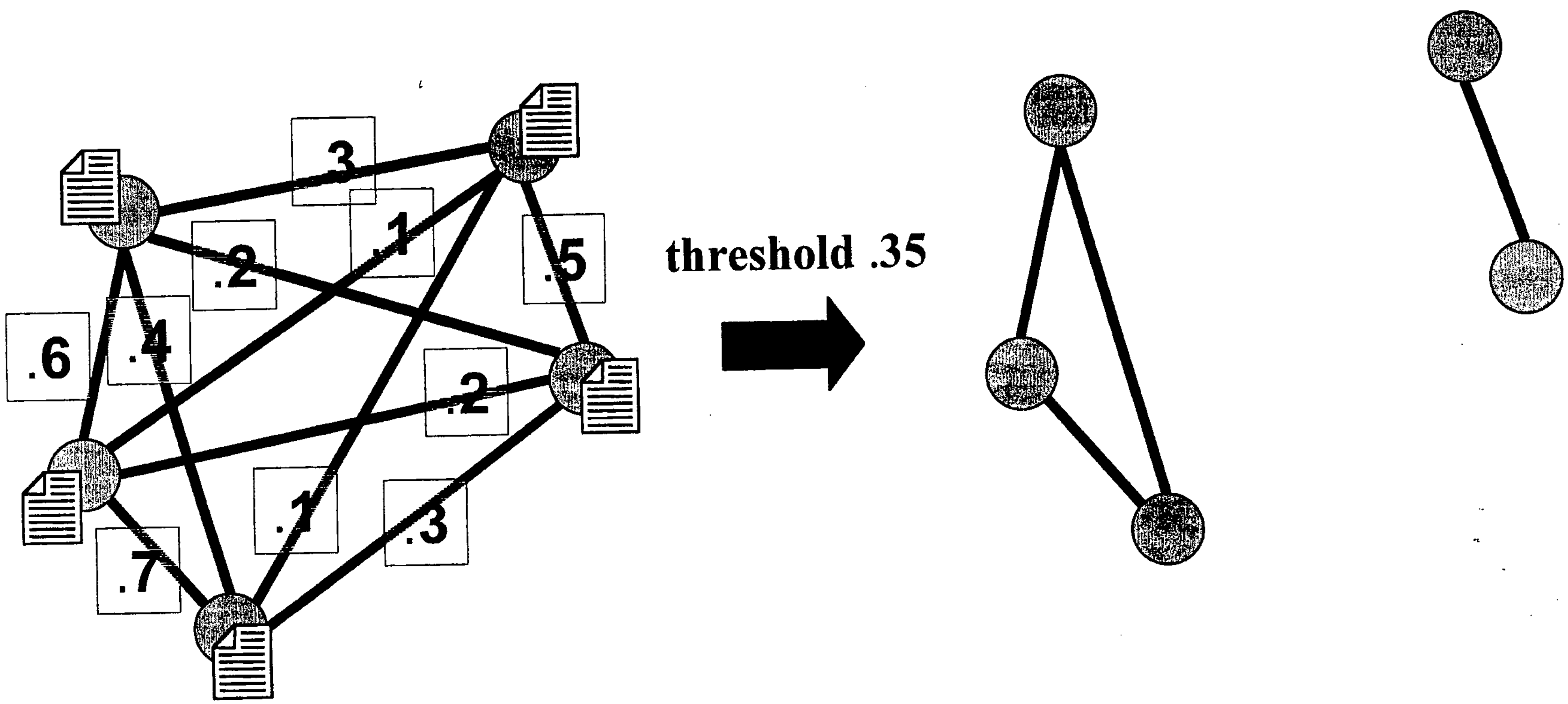
  vector of word frequencies
- Cosine similarity

  proportional to fraction of words in common
- Improvements

  stop lists, stemming, term weighting, thesaurus

Similarity graph models

- **Complete weighted graph**
  find similarities between all document pairs
- **Thresholded graph**

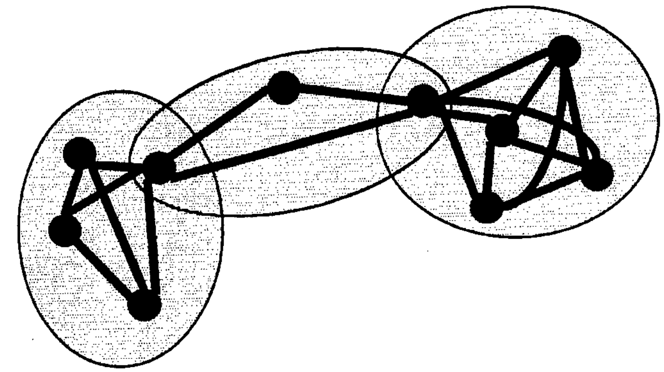


Accurate clusters

- **Clique cover of thresholded graph**

  accurate = guarantees minimum similarity between document in a cluster

  NP-hard



- **Cover by dense subgraphs**

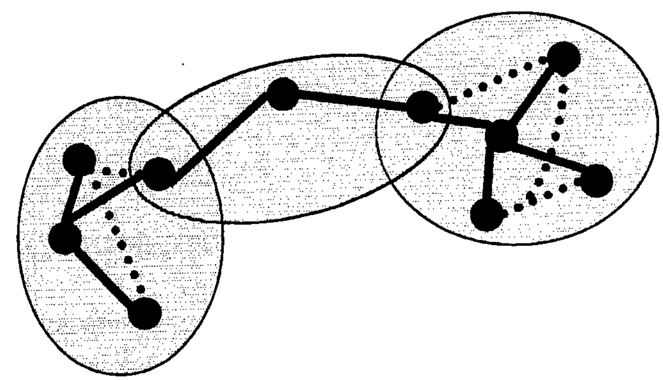
  approximates clique cover, sufficient for text document graphs

- **Cover by star subgraphs**

  constitutes dense subgraph cover

  star: a central vertex and adjacent vertices

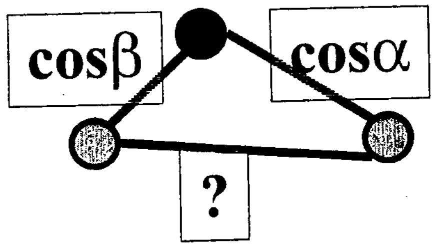
  Are star subgraphs dense?



Star subgraph density

- **Star subgraphs, found in a similarity graph, are dense**

star center



threshold σ

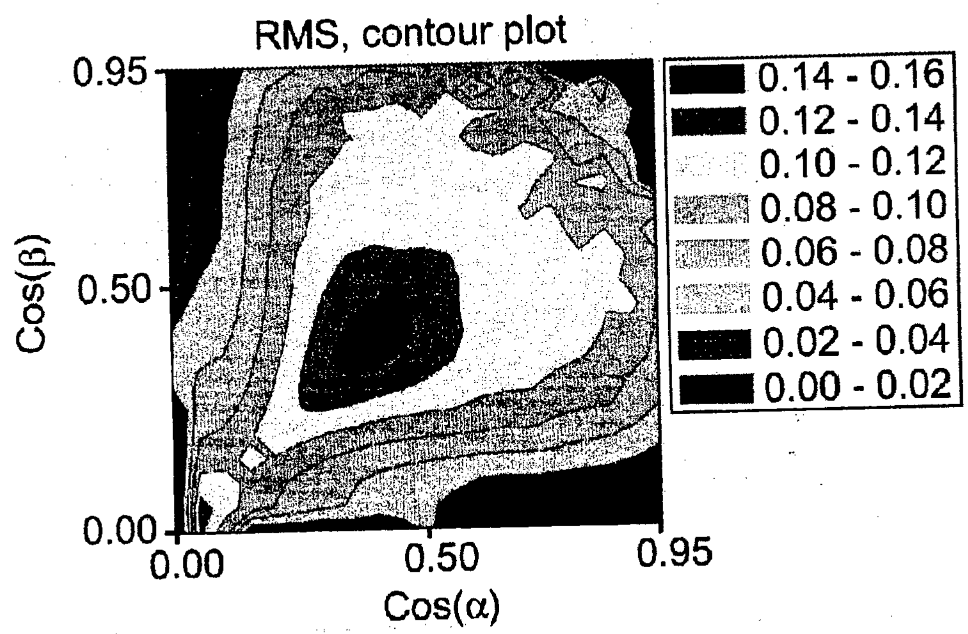
lower bound

$$\cos\alpha\cos\beta - \sin\alpha\sin\beta$$

expected value

$$\cos\alpha\cos\beta + \frac{\sigma}{1+\sigma}\sin\alpha\sin\beta$$

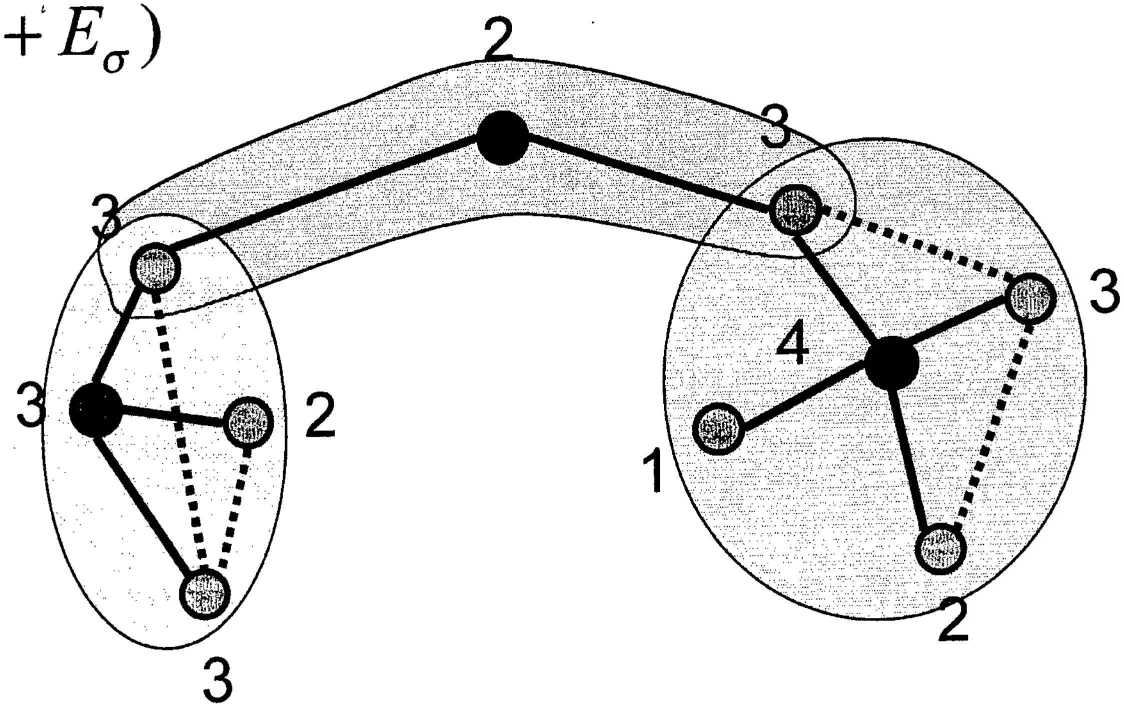
error



1/4/01

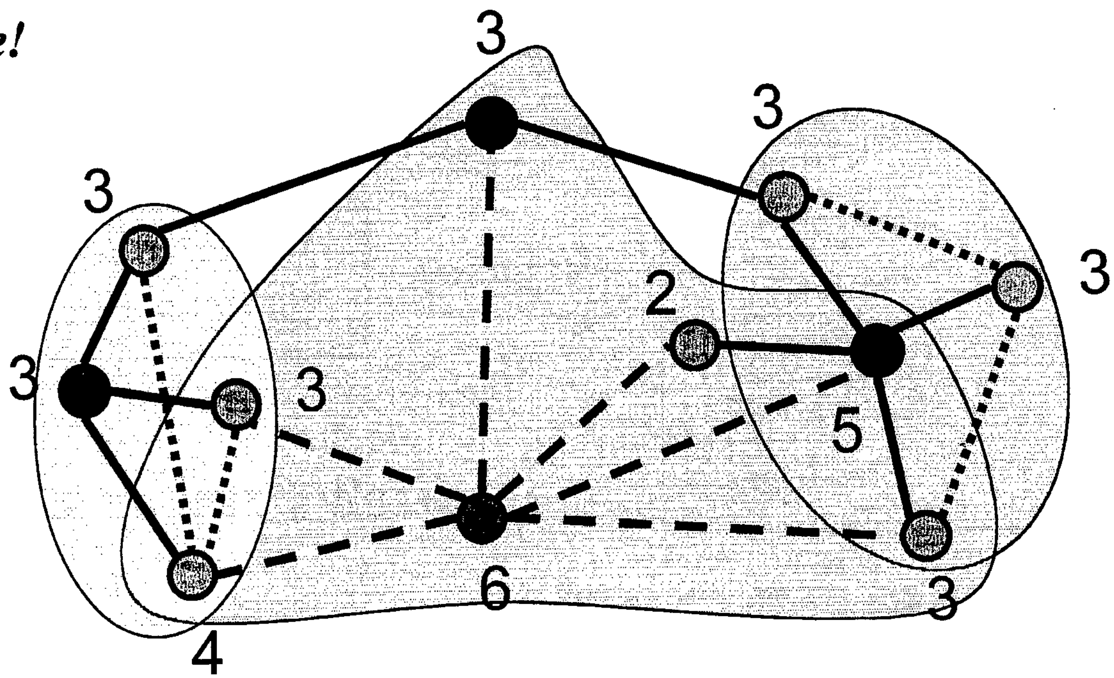
Star clustering algorithm

- thresholded graph
- find vertex degrees
- compute greedy cover
- time $\Theta(V + E_\sigma)$



Star clustering update

- Maintain star clustering
  - *1) vertex is a center if its degree is greater than the degree of any adjacent vertex; 2) no two centers are adjacent*
  - *only look at the vertices that violate this rule; specifically vertices that may become new star centers*
  - *changes may propagate!*



Star clustering update procedure

- **Finds the same clusters as star algorithm**

```
Update(α, L)
α.type ← satellite
α.degree ← 0
α.adj ← ∅
α.centers ← ∅
forall β in L
    α.degree ← α.degree + 1
    β.degree ← β.degree + 1
    Insert(β, α.adj)
    Insert(α, β.adj)
    if (β.type = center)
        Insert(β, α.centers)
    else
        β.inQ ← true
        Enqueue(β, Q)
    endif
endfor
α.inQ ← true
Enqueue(α, Q)
while (Q ≠ ∅)
    φ ← ExtractMax(Q)
    if (φ.centers = ∅)
        φ.type ← center
        forall β in φ.adj
            Insert(φ, β.centers)
        endfor
    else
        if (∀δ ∈ φ.centers, δ.degree < φ.degree)
            φ.type ← center
            forall β in φ.adj
                Insert(φ, β.centers)
            endfor
            forall δ in φ.centers
                δ.type ← satellite
                forall μ in δ.adj
                    Delete(δ, μ.centers)
                    if (μ.inQ = false)
                        μ.inQ ← true
                        Enqueue(μ, Q)
                    endif
                endfor
            endfor
        endif
    endif
    φ.inQ ← false
endwhile
```

add vertex to the queue, increment vertex degrees

extract vertex

no adjacent centers: new center

degree higher than any adjacent center: new center

Destroy adjacent stars, add to the queue

1/4/01

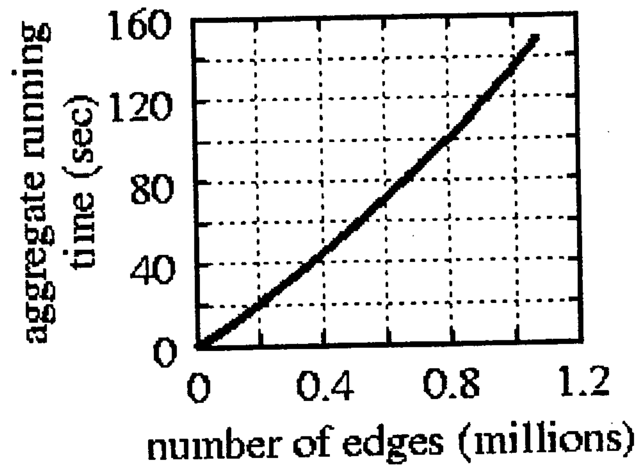
Update:running time

- **Aggregate running time**

  add V vertices

  worst case time $\Theta(V^2)$

  experimental time $\tilde{\Theta}(V + E_\sigma)$



- **Expected running time**

  random graph (V,p)

  expected size of star cover

$$1 + 2\log V / \log \frac{1}{1-p}$$

  expected time to add/delete vertex

$$O(Vp^2 \log^2 V / \log^2(1/(1-p)))$$

  aggregate running time (dense graphs)

$$O(V^2 \log^2 V)$$

Star clustering properties

- **Star algorithm**

  simple, efficient, overlapping clusters

- **Star clustering update**

  add/delete document

  computes the same clusters as the star algorithm

  efficient

- **Cluster hierarchy**

  clustering at all thresholds σ

  insert/delete edge

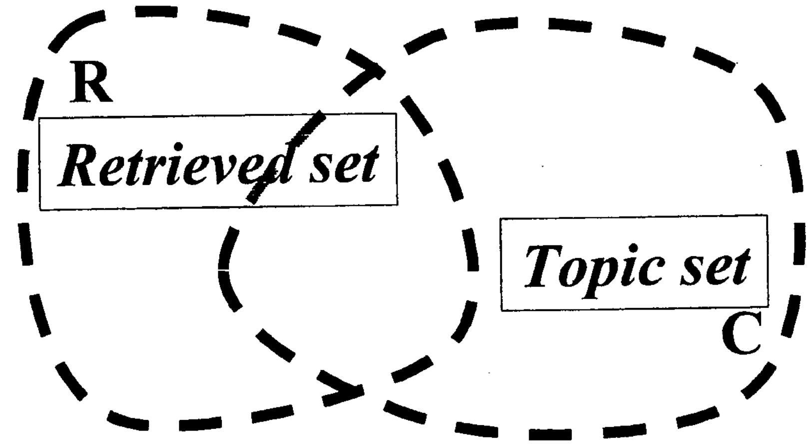
  time efficient

- **Goal**

  simple, efficient, <u>accurate clusters</u>, efficient updates, overlapping clusters, cluster hierarchy

Clustering evaluation

- **TREC filtering data**
  47 topics, 21694 documents
  relevant document sets



- **One cluster = one topic**
  find "best" cluster in a hierarchy
  compare to a topic document set

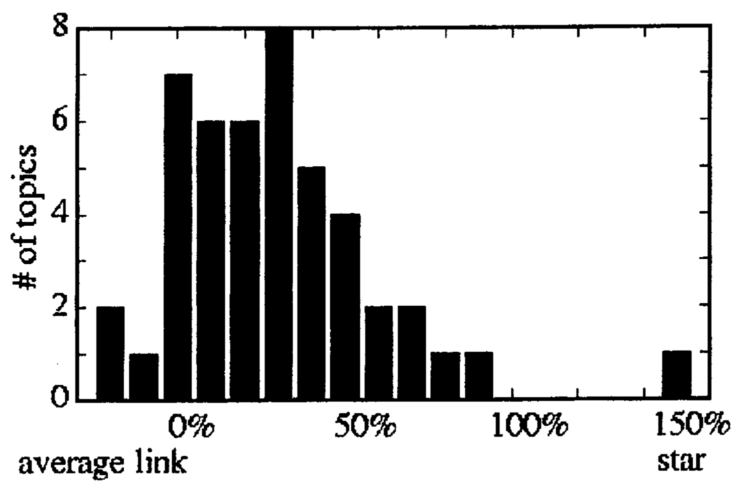
- **Measures**
  ***precision**: fraction of the retrieved documents that is relevant*
  ***recall**: fraction of relevant documents that is retrieved*

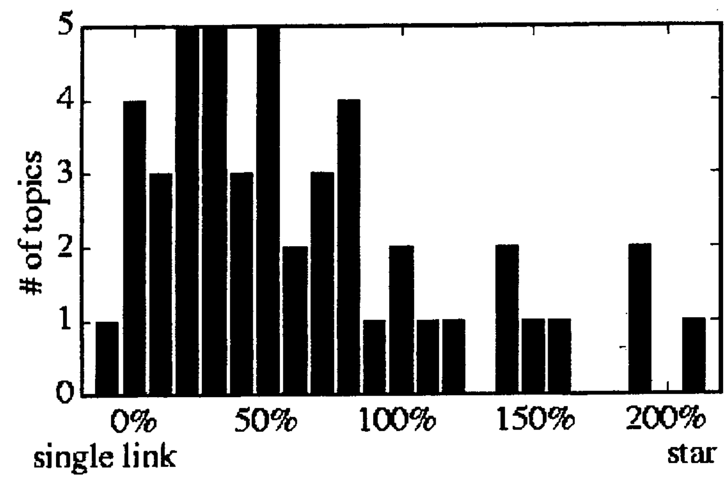
$$F = \frac{2|R \quad C|}{|R| + |C|} = \frac{2pr}{p+r}$$

Clustering Comparison

- Results (*F avg*)

star	*.42*	*(10.5%)*	*average link*	*.38*
		(40%)	*single link*	*.30*





1/4/01

Star Clustering: Final Words

- Simple
- Efficient

  bottleneck: similarity matrix computations

  random sampling approach [Aslam, Reiss, Rus '00]
- Accurate clusters
- Efficient updates

  similarity computations
- Overlapping clusters
- Cluster hierarchy

  time efficient

  storage: too many clusters

  maintain fewer hierarchy levels

Outline v3.0

- ~~Motivation~~

  Desperately need an information organization system!

- ~~Clustering Algorithm~~

  ~~*algorithm properties, design, evaluation*~~

  Simple, efficient, <u>accurate</u>, <u>dynamic</u>, parameterized clustering algorithm for information organization

- Applications

  retrieval, filtering, other…

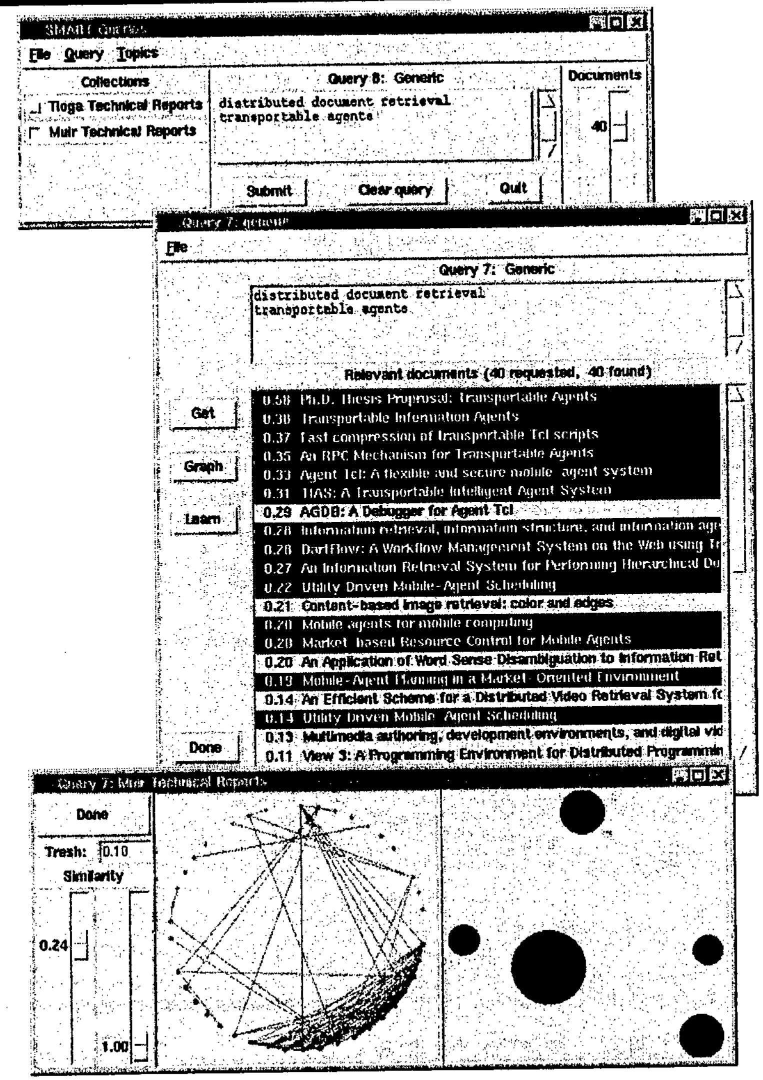
- Conclusions

  …and future work

1/4/01

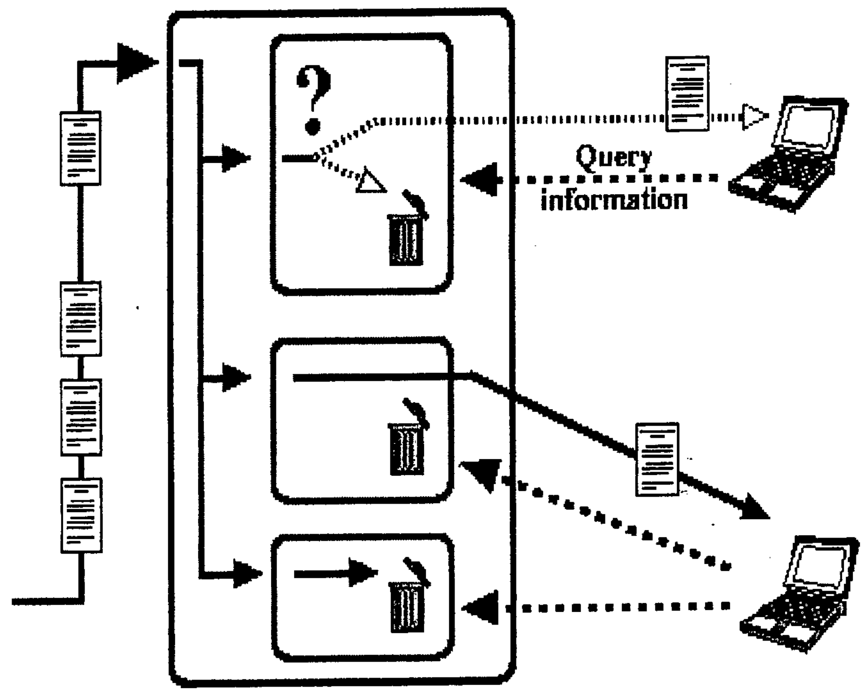
Organizing retrieval results

- **Ranked list retrieval**
  - *displays document titles*
- **Clustering the retrieved set**
  - *interactive threshold selection*
- **Visualization**
  - *document and cluster-based*



Filtering

- **Filtering profile**
  user specifies the profile
  new document is compared to the profile
- **Relevance feedback**
  on documents that pass the filter



Filtering Algorithms

- **Using star clustering algorithm**

  Find initial clustering using training data

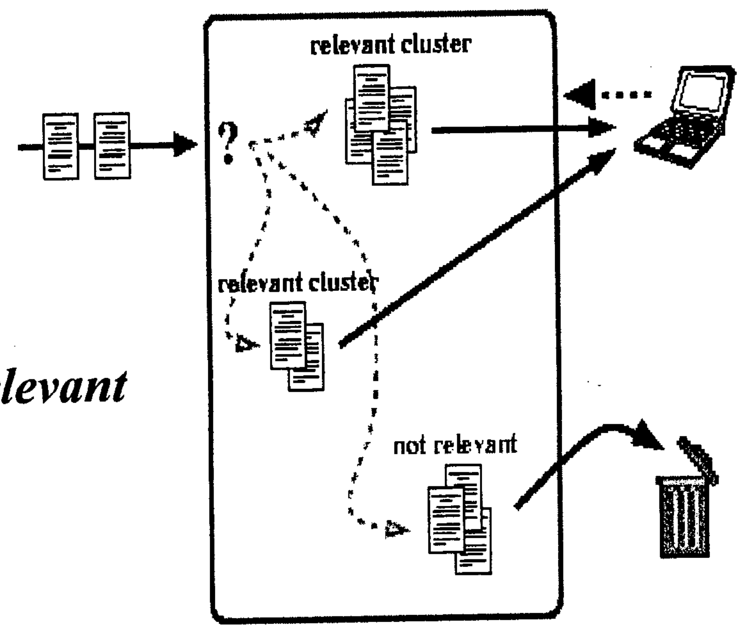
  clustering threshold, relevant clusters

  Add new document to clustering

  star clustering Update procedure

  Decide if new document is relevant

  relevant iff the adjacent star center is relevant



- **Using relevant document centroid**

  use training data to find cut-off threshold

- **Using one cluster**

  use training data to find clustering threshold, relevant cluster

Filtering:evaluation

- **Star filtering algorithm properties**

  use training data to find clustering threshold

  smaller relevant document set is used

- **Filtering algorithm comparison**

  TREC filtering collections: AP (50 topics), FBIS (47 topics); news (4 groups)

  compare retrieved set and topic relevant set using F measure

- **Results (F avg)**

FBIS:	*center .27*	*cluster .29*	*star .29*
AP:	*center .27*	*cluster .30*	*star .31*
News:	*center .89*	*cluster .91*	*star .91*

Filtering system implementation

- **User has no access to clusters**
  - *cannot take advantage of smaller relevance feedback set*
  - *automatic clustering threshold selection*
  - *careful not to retrieve too many documetns*

- **Interactive filtering system (persistent queries)**
  - *extension to organized retrieval system*
  - *user guided clustering threshold selection*
  - *filtering topic defined by clusters*
  - *easy topic exploration and query refinement*
  - *cluster center document summarizes clusters*
  - *relevance feedback on cluster centers only*
  - *with clustering organization can retrieve all relevant documents*

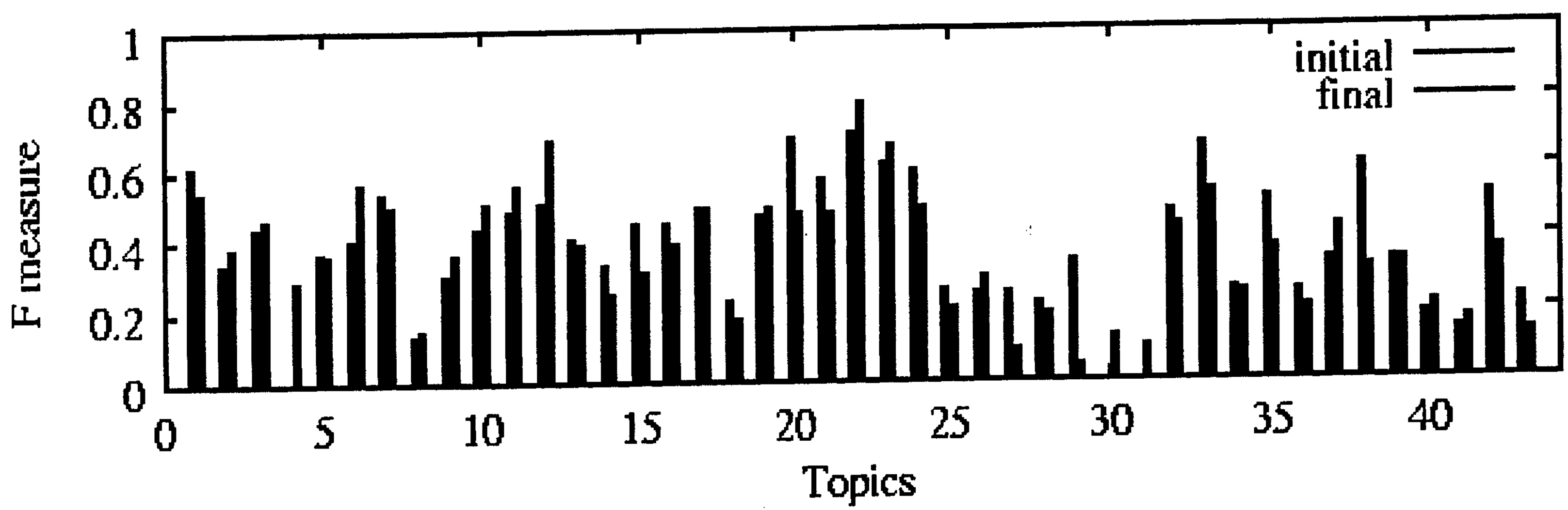
Persistent Query Performance

- **Model user behavior**

  select best matching cluster on training set
- **Simulate user input**

  make relevancy of new documents known to the algorithm
- **Collection**

  TREC AP filtering collection (50 topics)



Outline v4.0

- ~~Motivation~~

  Desperately need an information organization system!

- ~~Clustering Algorithm~~

  Simple, efficient, accurate, dynamic, parameterized clustering algorithm for information organization

- ~~Applications~~

  ~~*retrieval, filtering, other…*~~

  Interactive applications: organized retrieval, filtering (persistent queries)

- Conclusions

  …and future work

Conclusions

- Star clustering algorithm

  time and space efficient

  accurate clustering

  efficient clustering update

  efficient cluster hierarchy

- Information organization applications

  organized retrieval application

  visualization algorithms

  three filtering algorithms

  method for automatic threshold selection

  persistent queries: interactive filtering application

  star filtering algorithm: maintains topic accuracy

Future Directions

- **Scalability**
  hierarchy, clustering updates
- **Browsing**
  "natural" cluster hierarchy
- **Filtering**
  threshold selection with minimal training data
- **Distributed systems**
- **Visualization**
- **Topic summarization**

Sample Questions (FAQ)

- **Will it be raining again this weekend?**

  I do not know.
- **Who I should thank for the cookies?**

  Anna. Thanks!
- **I still don't understand the graph on slide 16. How is it related to star clustering?**

  It is a very interesting question - I am glad you have asked...
- **What is the value of Hubble constant?**

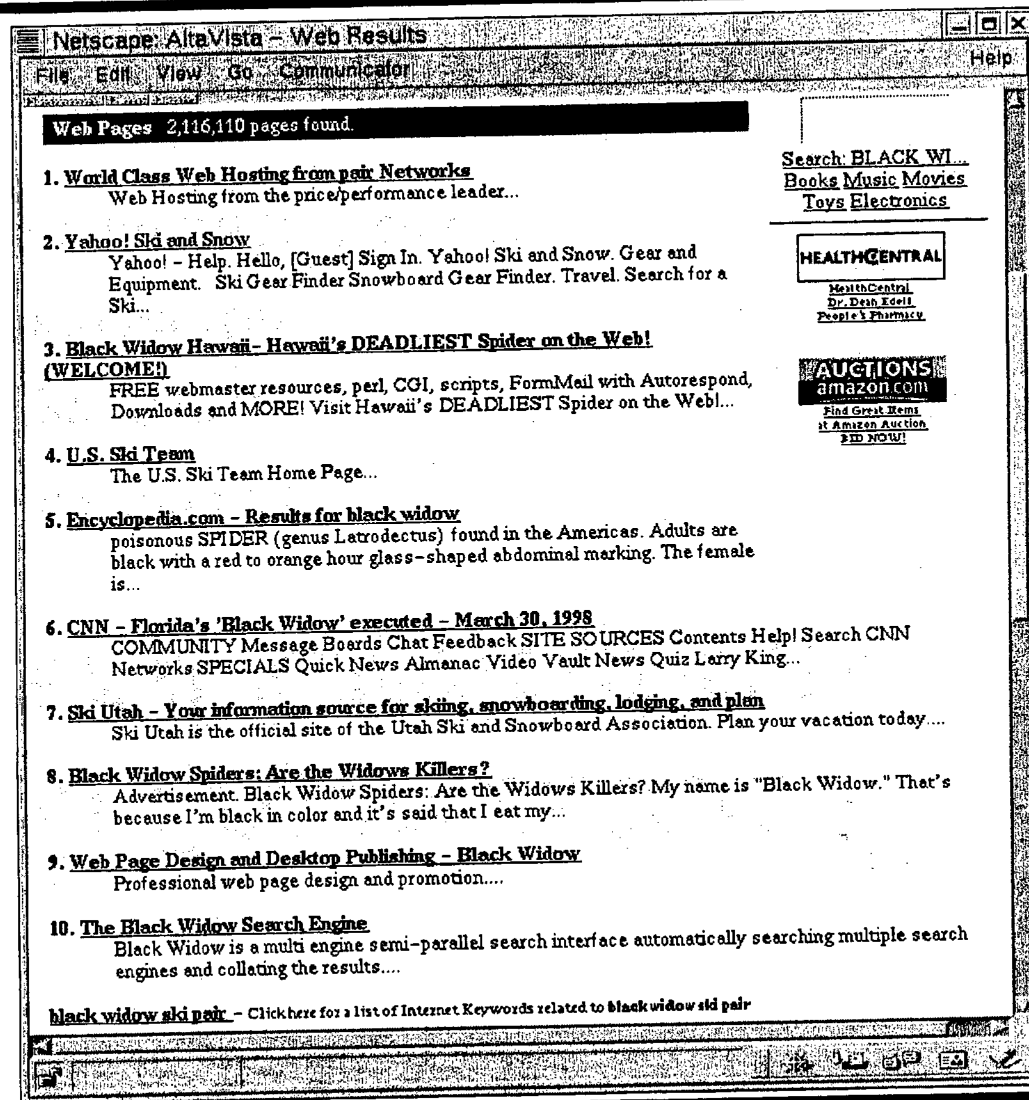
  .1 to 80
- **Why all the pictures are fuzzy?**

Information Organization Algorithms and Applications

Jay Aslam
Katya Pelekhov
Daniela Rus
Dartmouth Computer Science

Misunderstood query

- **Query**
  "black widow ski pair"
- **Meaning**
  I am looking for skis called "black widows"
- **Search engine**
  spiders?
  au pair?
  ...?
  skiing?
- **Ranked list**
  all topics above



Information Organization

- **Prevailing approach**

  most commercial search engines

  find everything remotely relevant

  frustrated user

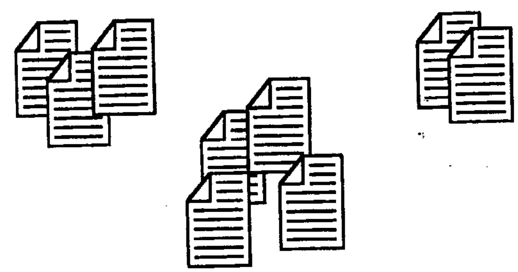
- **Organization approach**

  NorthernLight, many research systems

  organize retrieved document by topics

  manageable amount of information

  helps user to navigate and narrow the request



Other applications

- **Information overload is not limited to search**
- **Retrieval applications**

  Consider only text collections

  Classify by collection dynamics and user's interest

  Not consider nature of the documents (news events, stocks, etc.)

	searching	browsing	filtering
collection	stable	stable	changing
user's interests	changing	broad	stable

- **Hybrid applications**

  i.e., dynamic collection, changing user interests

  modern information system trend

Organizing system

- **Efficient**
- **Find accurate topics**

  topics that naturally occur in a collection
- **Update topics (add/delete documents)**

  while maintaining topic accuracy
- **Find topic hierarchies**
- **Compute topic summaries**

  readable (user)

  internal representation (programs)
- **Presentation**

  summarization and visualization of information

Methods for organization:

manual, classification, <u>clustering</u>

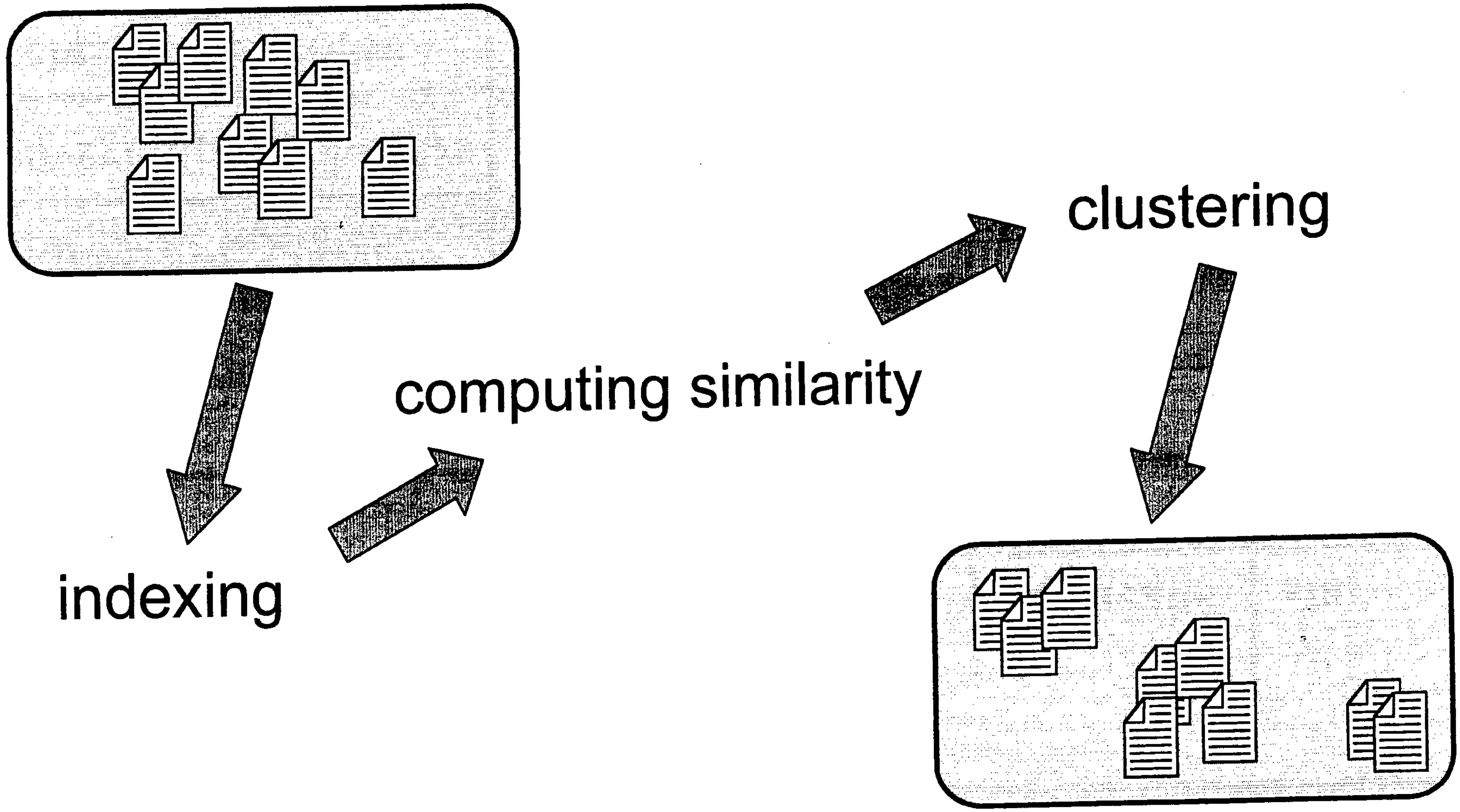
Related Work

- **Applications**
  - *Organizing retrieval results*
    - Hearst & Pedersen, 1996; NorthernLight; others
  - *Browsing*
    - Scatter/Gather, 1992; Yahoo!
  - *Filtering*
    - classification algorithms; Eichmann et al. 1998; others
  - *Other applications*
    - distributed retrieval, data mining, topic detection and tracking, ...
- **Methods**
  - *Cluster hypothesis*
    - similar documents are relevant to the same requests (van Rijsbergen 1971)
  - *Clustering algorithms*
    - single link, average link, k-means, others

Clustering Properties

	single link	average link	*K*-means	star
simple	P	P	P	P
efficient	P	P	P	P
accurate clusters		P		P
efficient updates	P			P
overlapping clusters			P	P
cluster hierarchy	P	P	P	P

Clustering system design



Vector Space model

- "Bag of words" document representation

  vector of word frequencies

- Cosine similarity

  proportional to fraction of words in common

- Improvements

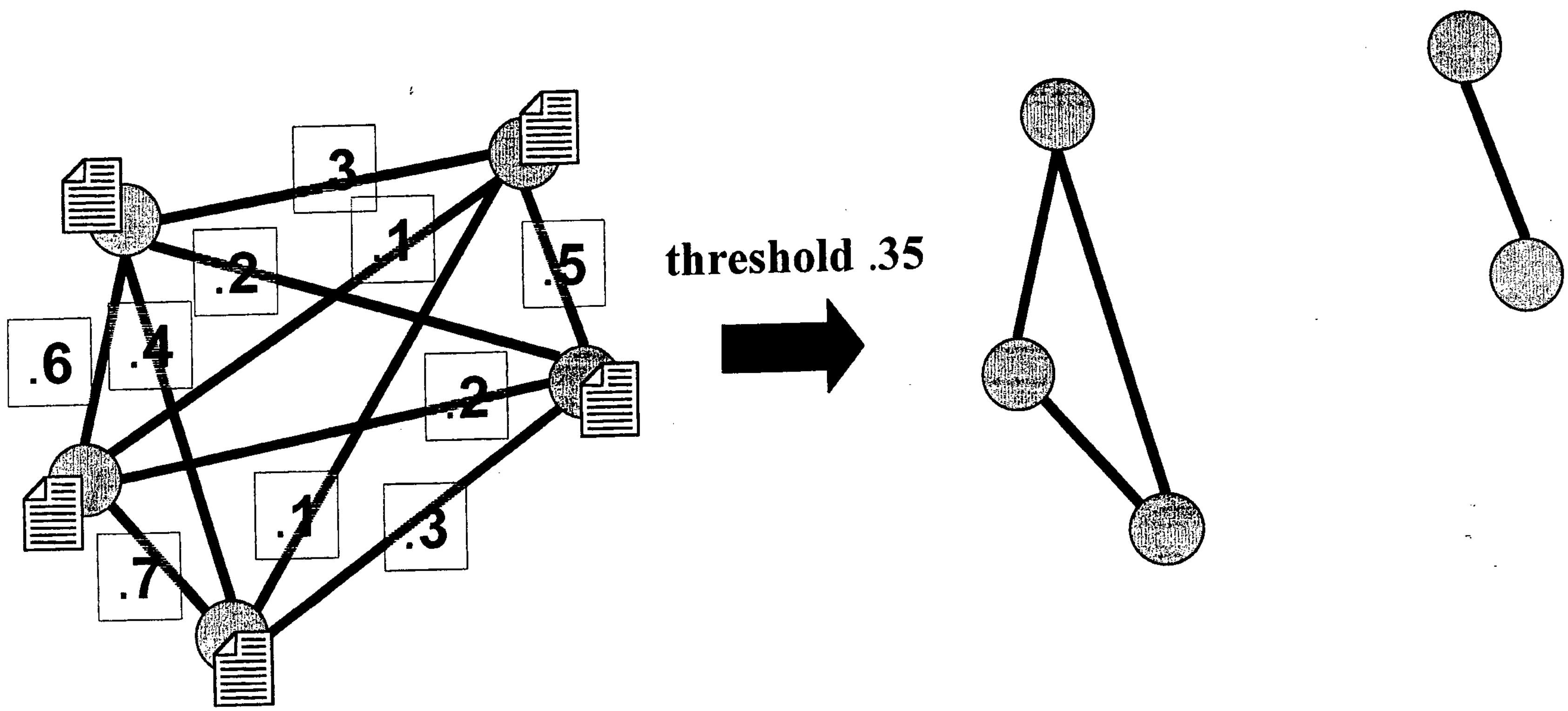
  stop lists, stemming, term weighting, thesaurus

Similarity graph models

- **Complete weighted graph**

  find similarities between all document pairs

- **Thresholded graph**

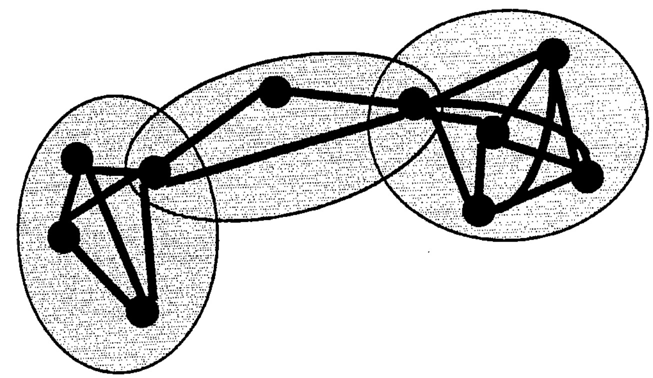


Accurate clusters

- **Clique cover of thresholded graph**

  accurate = guarantees minimum similarity between document in a cluster

  NP-hard



- **Cover by dense subgraphs**

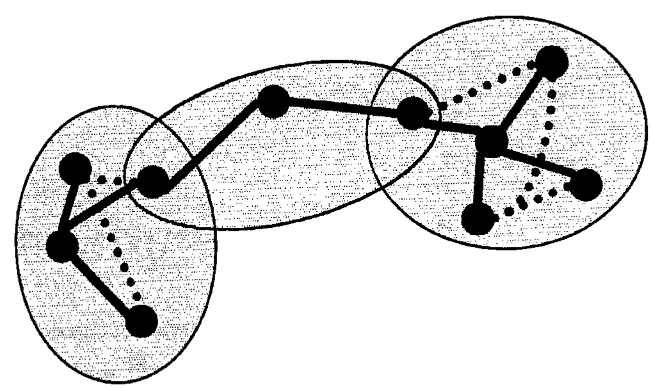
  approximates clique cover, sufficient for text document graphs

- **Cover by star subgraphs**

  constitutes dense subgraph cover

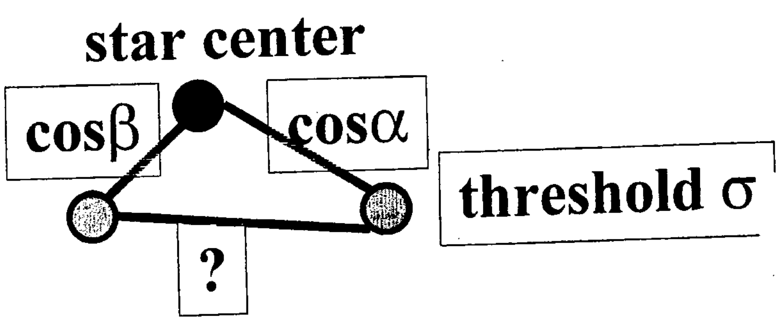
  star: a central vertex and adjacent vertices

  Are star subgraphs dense?



Star subgraph density

- **Star subgraphs, found in a similarity graph, are dense**



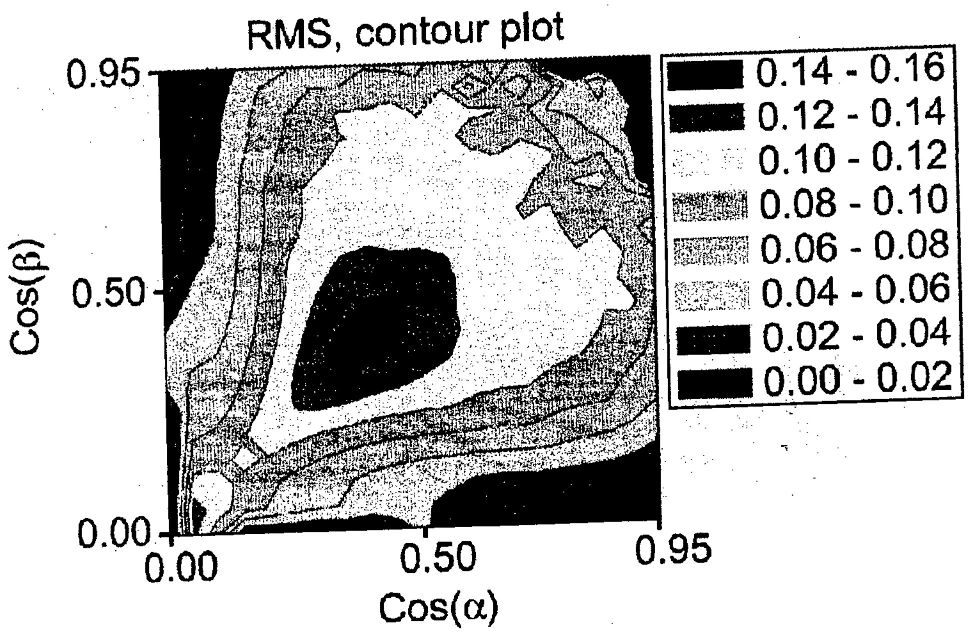
lower bound

$$\cos\alpha\cos\beta - \sin\alpha\sin\beta$$

expected value

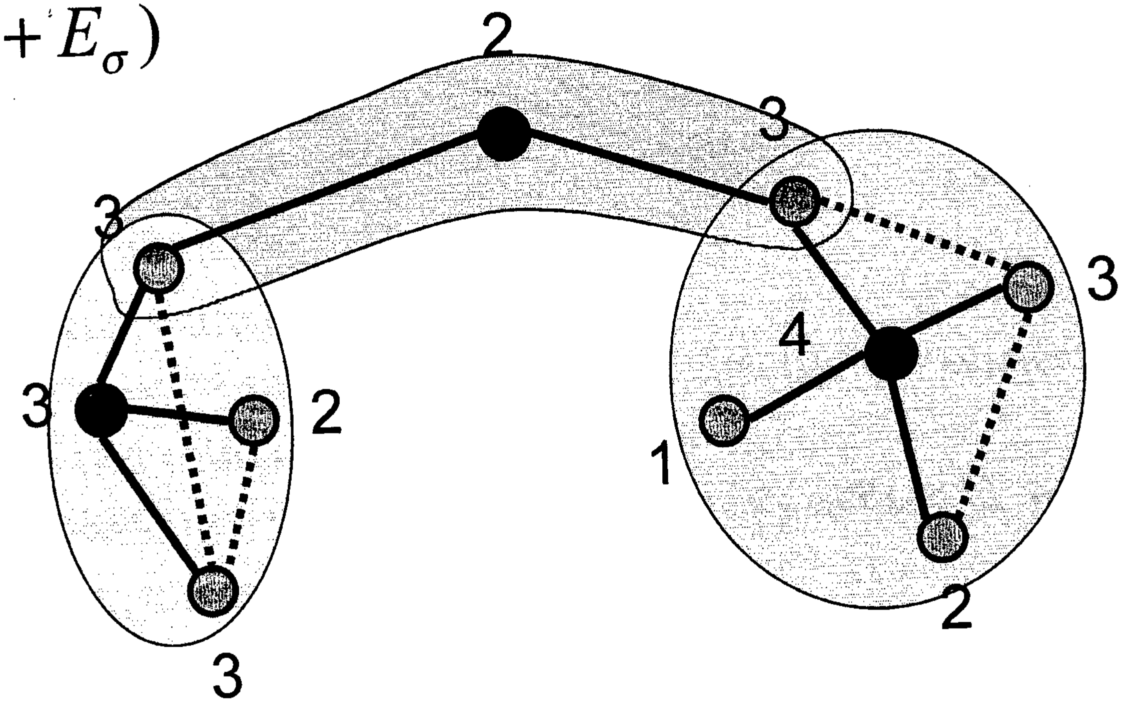
$$\cos\alpha\cos\beta + \frac{\sigma}{1+\sigma}\sin\alpha\sin\beta$$

error



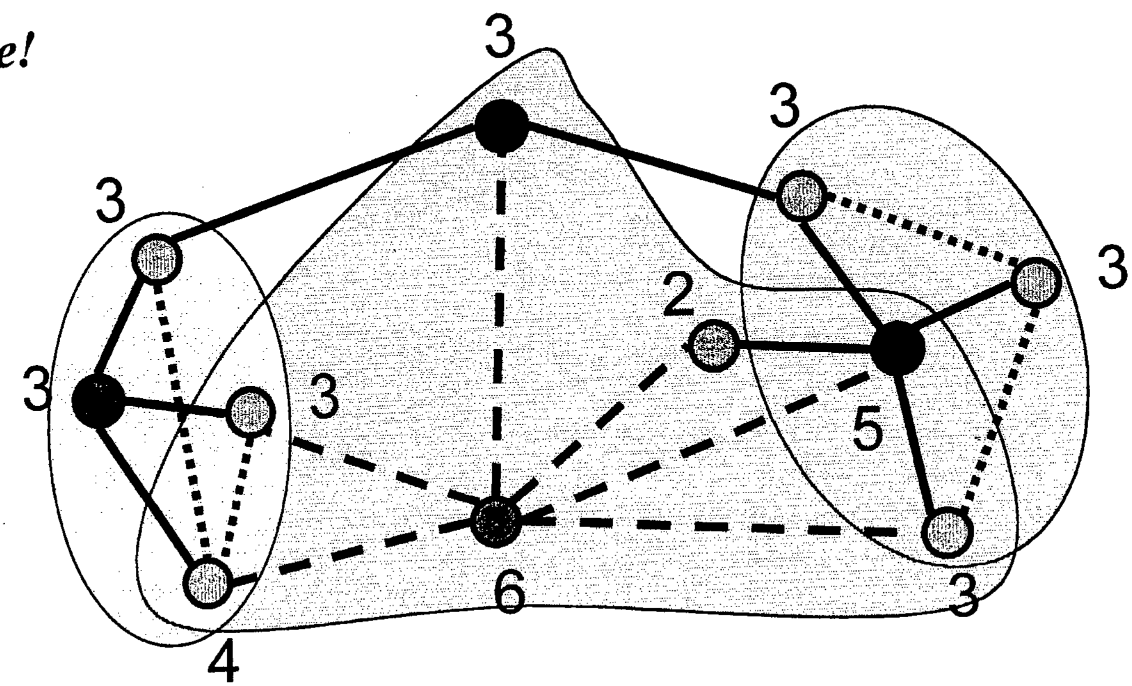
Star clustering algorithm

- thresholded graph
- find vertex degrees
- compute greedy cover
- time $\Theta(V + E_\sigma)$



Star clustering update

- **Maintain star clustering**
  - *1) vertex is a center if its degree is greater than the degree of any adjacent vertex; 2) no two centers are adjacent*
  - *only look at the vertices that violate this rule; specifically vertices that may become new star centers*
  - *changes may propagate!*



Star clustering update procedure

- Finds the same clusters as star algorithm

add vertex to the queue, increment vertex degrees

extract vertex

no adjacent centers: new center

degree higher than any adjacent center: new center

Destroy adjacent stars, add to the queue

```
UPDATE(α, L)
  α.type ← satellite
  α.degree ← 0
  α.adj ← ∅
  α.centers ← ∅
  forall β in L
     α.degree ← α.degree + 1
     β.degree ← β.degree + 1
     INSERT(β, α.adj)
     INSERT(α, β.adj)
     if (β.type = center)
        INSERT(β, α.centers)
     else
        β.inQ ← true
        ENQUEUE(β, Q)
     endif
  endfor
  α.inQ ← true
  ENQUEUE(α, Q)
  while (Q ≠ ∅)
     φ ← EXTRACTMAX(Q)
     if (φ.centers = ∅)
        φ.type ← center
        forall β in φ.adj
           INSERT(φ, β.centers)
        endfor
     else
        if (∀δ ∈ φ.centers, δ.degree < φ.degree)
           φ.type ← center
           forall β in φ.adj
              INSERT(φ, β.centers)
           endfor
           forall δ in φ.centers
              δ.type ← satellite
              forall μ in δ.adj
                 DELETE(δ, μ.centers)
                 if (μ.inQ = false)
                    μ.inQ ← true
                    ENQUEUE(μ, Q)
                 endif
              endfor
           endfor
        endif
     endif
     φ.inQ ← false
  endwhile
```

Star clustering properties

- **Star algorithm**

  simple, efficient, overlapping clusters

- **Star clustering update**

  add/delete document

  computes the same clusters as the star algorithm

  efficient

- **Cluster hierarchy**

  clustering at all thresholds σ

  insert/delete edge

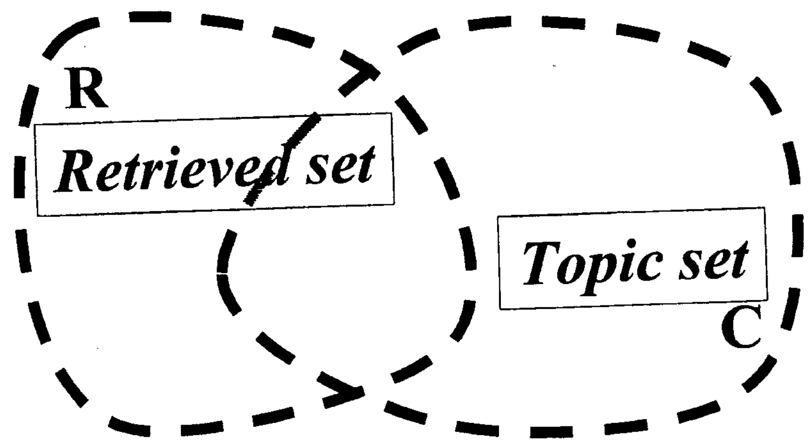
  time efficient

- **Goal**

  simple, efficient, <u>accurate clusters</u>, efficient updates, overlapping clusters, cluster hierarchy

Clustering evaluation

- **TREC filtering data**
  - ***47 topics, 21694 documents***
  - ***relevant document sets***
- **One cluster = one topic**
  - ***find "best" cluster in a hierarchy***
  - ***compare to a topic document set***



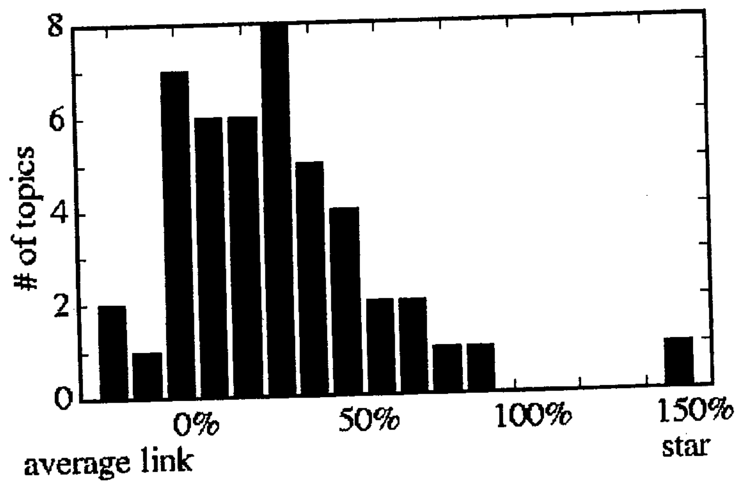
- **Measures**
  - ***precision: fraction of the retrieved documents that is relevant***
  - ***recall: fraction of relevant documents that is retrieved***

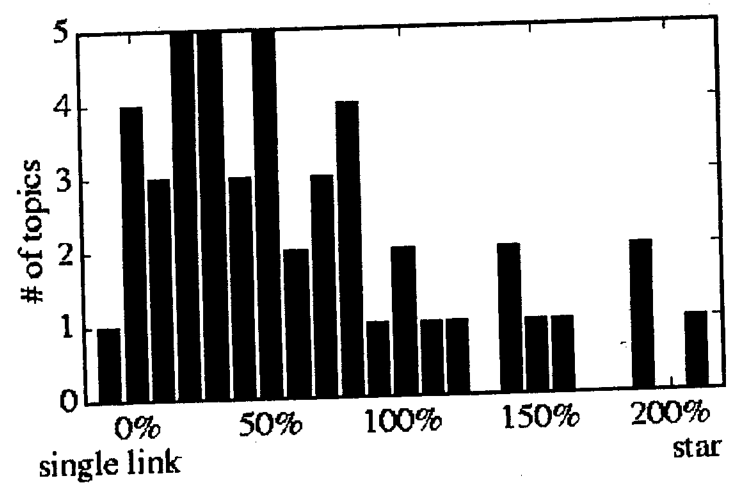
$$F = \frac{2|R \cap C|}{|R| + |C|} = \frac{2pr}{p+r}$$

Clustering Comparison

- Results (*F avg*)

star	*.42*	*(10.5%)*	*average link*	*.38*
		(40%)	*single link*	*.30*





Star Clustering: Final Words

- **Simple**
- **Efficient**

  bottleneck: similarity matrix computations

  random sampling approach [Aslam, Reiss, Rus '00]
- **Accurate clusters**
- **Efficient updates**

  similarity computations
- **Overlapping clusters**
- **Cluster hierarchy**

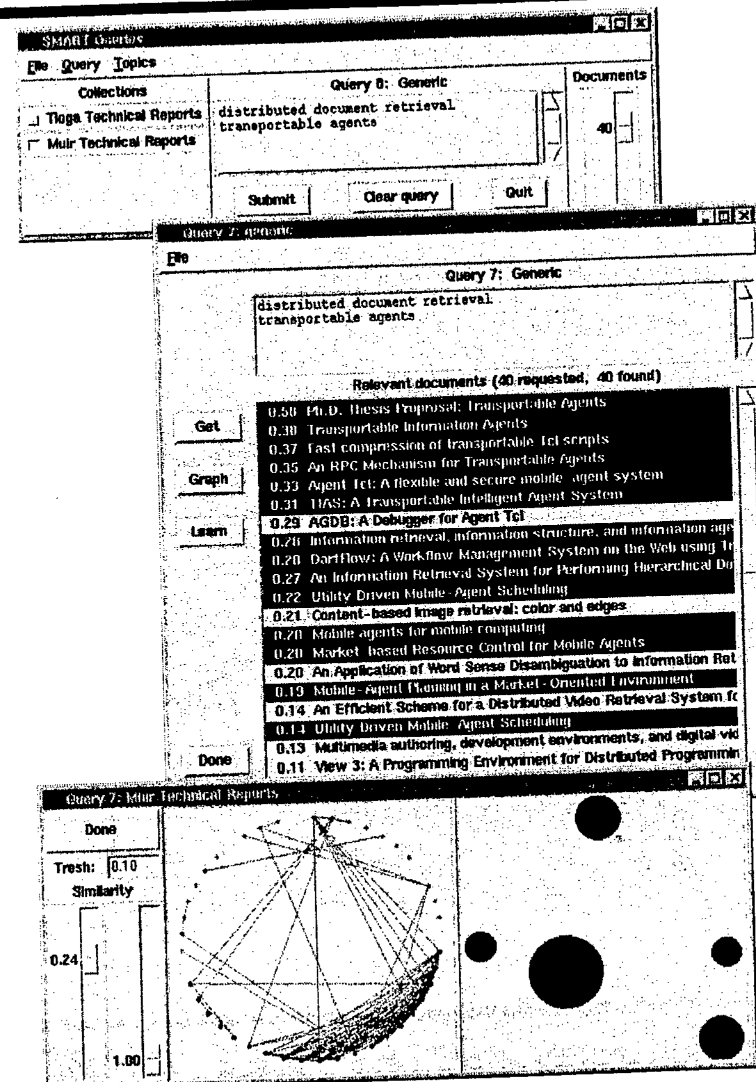
  time efficient

  storage: too many clusters

  maintain fewer hierarchy levels

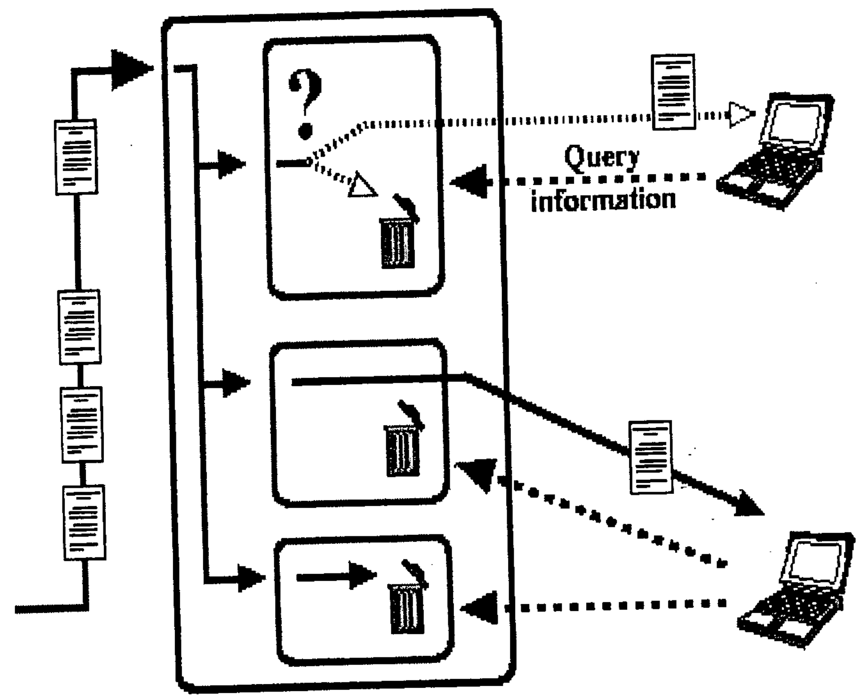
Organizing retrieval results

- **Ranked list retrieval**
  - ***displays document titles***
- **Clustering the retrieved set**
  - ***interactive threshold selection***
- **Visualization**
  - ***document and cluster-based***



Filtering

- **Filtering profile**
  user specifies the profile
  new document is compared to the profile
- **Relevance feedback**
  on documents that pass the filter



Filtering Algorithms

- **Using star clustering algorithm**

  Find initial clustering using training data

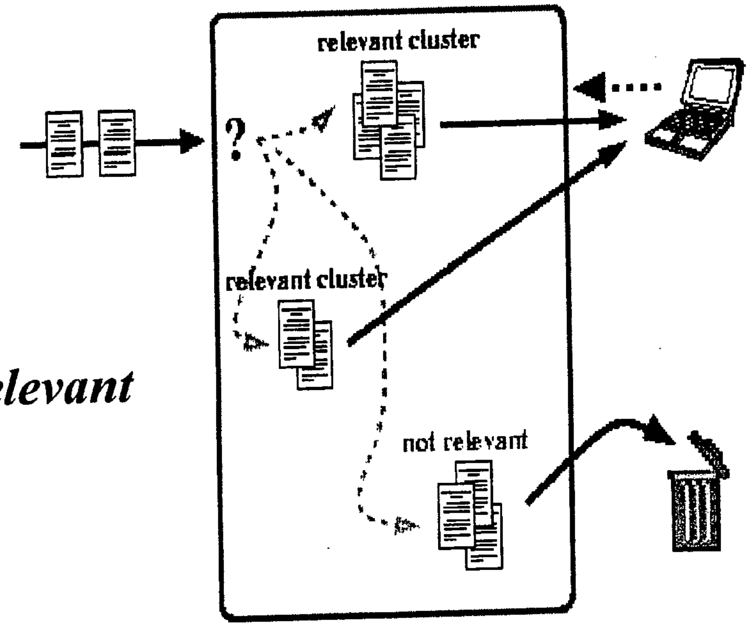
  clustering threshold, relevant clusters

  Add new document to clustering

  star clustering Update procedure

  Decide if new document is relevant

  relevant iff the adjacent star center is relevant



- **Using relevant document centroid**

  use training data to find cut-off threshold

- **Using one cluster**

  use training data to find clustering threshold, relevant cluster

Filtering:evaluation

- **Star filtering algorithm properties**

  use training data to find clustering threshold

  smaller relevant document set is used

- **Filtering algorithm comparison**

  TREC filtering collections: AP (50 topics), FBIS (47 topics); news (4 groups)

  compare retrieved set and topic relevant set using F measure

- **Results (F avg)**

FBIS:	*center .27*	*cluster .29*	*star .29*
AP:	*center .27*	*cluster .30*	*star .31*
News:	*center .89*	*cluster .91*	*star .91*

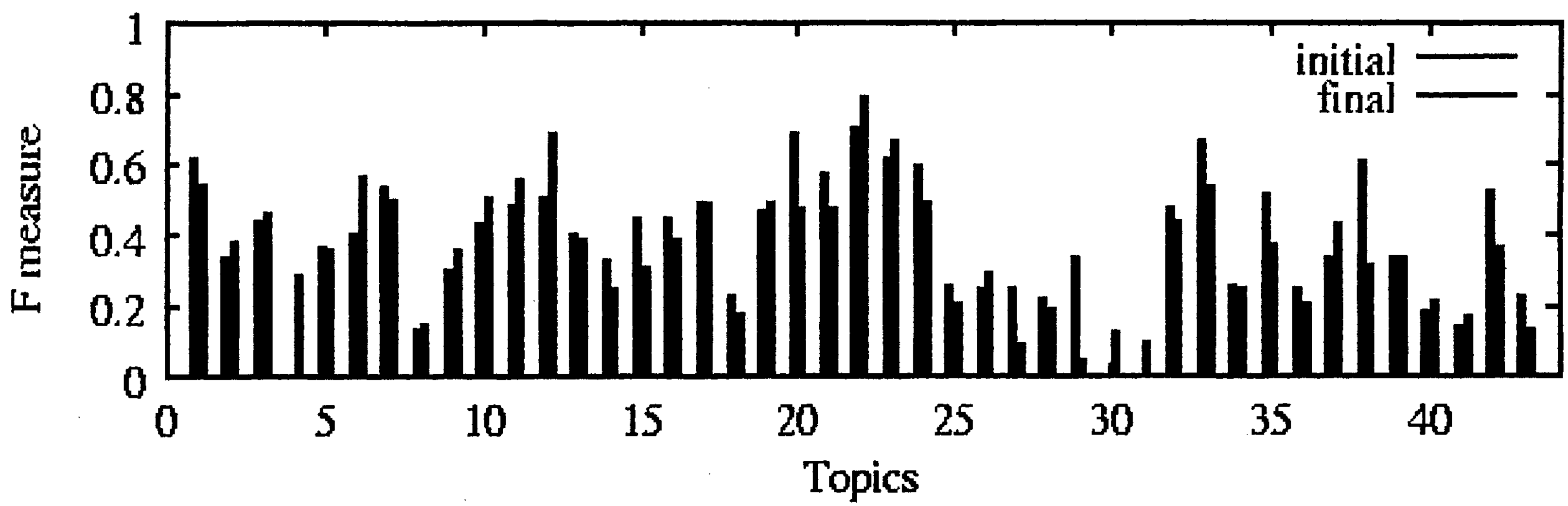
Filtering system implementation

- **User has no access to clusters**
  - *cannot take advantage of smaller relevance feedback set*
  - *automatic clustering threshold selection*
  - *careful not to retrieve too many documetns*

- **Interactive filtering system (persistent queries)**
  - *extension to organized retrieval system*
  - *user guided clustering threshold selection*
  - *filtering topic defined by clusters*
  - *easy topic exploration and query refinement*
  - *cluster center document summarizes clusters*
  - *relevance feedback on cluster centers only*
  - *with clustering organization can retrieve all relevant documents*

Persistent Query Performance

- **Model user behavior**
  select best matching cluster on training set
- **Simulate user input**
  make relevancy of new documents known to the algorithm
- **Collection**
  TREC AP filtering collection (50 topics)



Conclusions

- **Star clustering algorithm**

  time and space efficient

  accurate clustering

  efficient clustering update

  efficient cluster hierarchy

- **Information organization applications**

  organized retrieval application

  visualization algorithms

  three filtering algorithms

  method for automatic threshold selection

  persistent queries: interactive filtering application

  star filtering algorithm: maintains topic accuracy

©U.S. GOVERNMENT PRINTING OFFICE: 2001-610-055-10076

MISSION
OF
AFRL/INFORMATION DIRECTORATE (IF)

The advancement and application of Information Systems Science and Technology to meet Air Force unique requirements for Information Dominance and its transition to aerospace systems to meet Air Force needs.